# Blend Switching User Guide

Ruslan Kurdyumov

October 26, 2011

## Contents

## Summary:

We have implemented real-time switching between blend filters for ITMX. The switching relies on two identical filter banks, the CURR and NEXT bank, each of which contains the possible blend filters. We use a C code block in Simulink to output a ramping value between 0 and 1, which allows us to smoothly ramp from the output of the CURR bank to the output of the NEXT bank. A Perl script takes care of turning on and off the proper filter modules and setting the value of an EPICS variable which controls the operation of the C block.

## FAQ:

*Why use two filter banks instead of switching between filter modules in one bank?*

There is no way to smoothly switch between filter modules in one bank.

*Why switch back to the CURR bank? Can't you just alternate which bank is the CURR bank?*

You can and it would make the switching faster. The basic answer – it's easier to display in MEDM. The highlighting of the current blend and the display of the current blend output rely on the fact that the CURR bank is fixed – it's always bank 1. You can switch once and design other displays, but we didn't think they would be as intuitive.

*What happens if I try to switch blends while switching is active?*

The Perl script called by MEDM will ignore any switch requests that include an actively switching blend. Therefore, if CPSX is switching and you choose to SWITCH_ALL to a different blend, the SWITCH_ALL request will be ignored even though 5 of the 6 DOFs are free to switch.

*What happens if I accidentally switch to my current blend?*

The current blend will be loaded into the NEXT bank and you will go through the switching process. Your blend will stay the same throughout. We chose to allow this, rather than ignoring the request, to keep the Perl script as simple as possible.

*What happens if I switch to an empty blend?*

The Perl script will not allow you to switch to an empty blend by checking the name field of the module to which the user wants to switch.

*Is the blend ramping time fixed?*

Yes, we have hardcoded the ramping time to 5 seconds for simplicity. The time can be changed in the BLENDMIXER.c file.

*In the Simulink model of the blend filters, why do you have a test point and an EPICS output for each blend output?*

You need the test point to sample the blend output at the model rate and you need the EPICS output to display the blend output in MEDM.

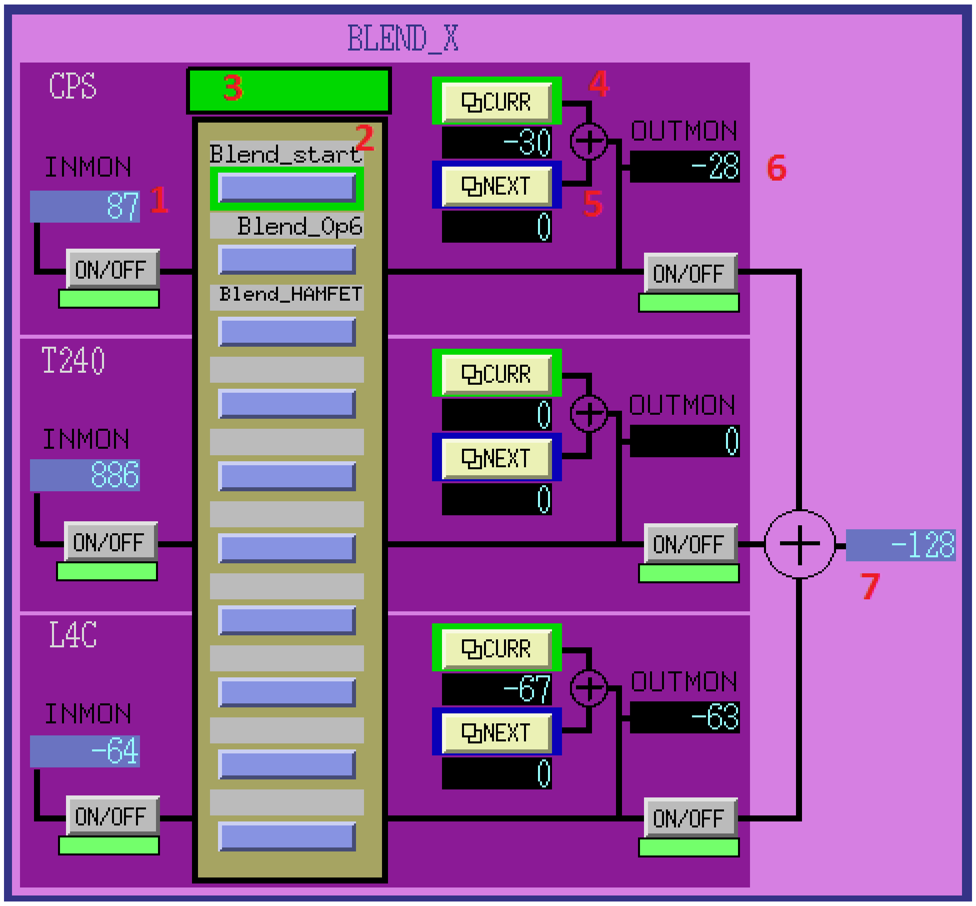
*Why did you append a 1 to the CURR filter bank (i.e. CPSX1 rather than CPSX) in Simulink?*

We wanted the mixed output of the blend to have the name CPSX_OUT for backwards compatibility in data gathering. If we had called the CURR bank CPSX, CPSX_OUT would not correspond to the mixed blend output.

## Operation:

### The Blend Screen

At the lowest level, the user can switch between blends for a given degree of freedom. Below, we have the MEDM screen illustrating the state of the blend filters for the Stage 1 X DOF, which blends 3 sensors: CPS, T240, and L4C:



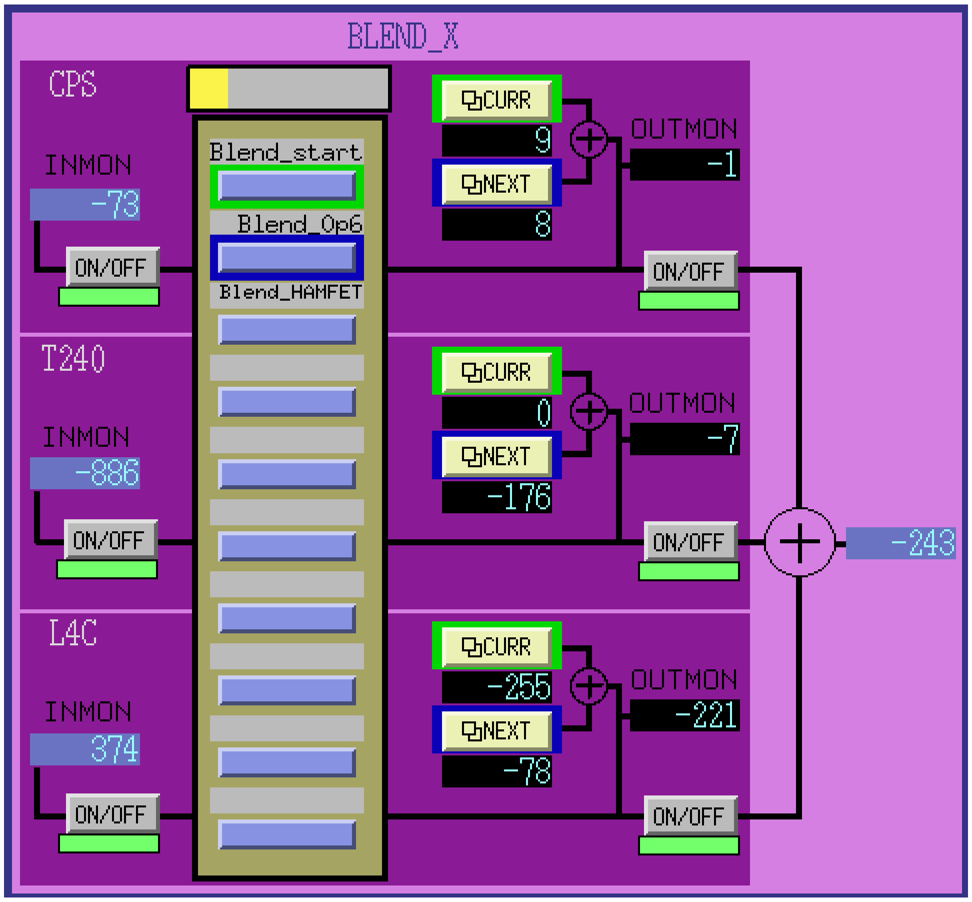
The important parts of the screen are numbered in red and explained below:

1. The input signal for the sensor to be blended. In this case – the CPS X signal.
2. The "blend bank", a filter bank which illustrates the possible blends we can choose from, and highlights the currently active blend in green. We can choose another blend by clicking on its button. In this case, the "Blend_start" blend is currently active, and we can click on "Blend_0p6" or "Blend_HAMFET" to switch blends.
3. The progress bar, which visually indicates how far along a blend switch we are. In this case, the bar is solid green, indicating that only one blend is active. During switching, the bar will progressively light up yellow indicator bars tracking how far along the switch we are.

4. The “CURR” blend output signal. This signal displays the output of the currently active blend (the one highlighted green in the “blend bank”). In this case, this is the “Blend_start” blend.[1]
5. The “NEXT” blend output signal. This signal displays the output of the blend we are switching to. In this case, we aren’t switching, so the output signal is 0. If we were switching, one of the blends in the “blend bank” would be highlighted blue and the “NEXT” signal would display the output of that blend.
6. The mixed blend output, which is the weighted sum of the CURR and NEXT outputs, depending on the switching progress. In this case, we are not switching, so the CURR output is weighted 1 and the NEXT output is weighted 0. At this point, all the sensors are in comparable units.
7. The supersensor signal, which adds the mixed blend outputs to produce an overall signal for the given DOF.

## Switching Between Blends

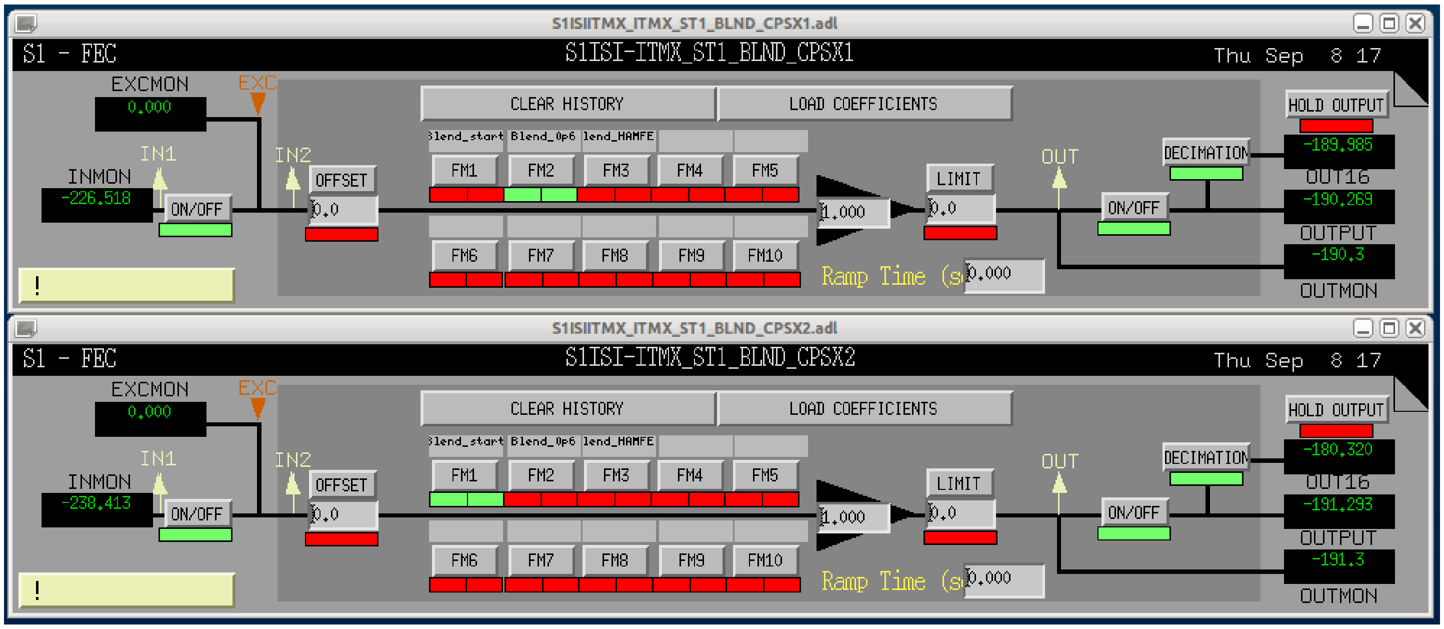
To switch to a different blend, you click the button of the desired blend. A screen like the one below will appear:



[1] Note that the T240 CURR output is 0 because the “Blend_start” blend filter happens to ignore this sensor.

During switching, the progress bar will update depending on how far along the switch you are. It will go back to solid green when switching is complete. The mixed blend output will also track the weighted sum of the CURR and NEXT blend outputs. In this case, we are switching from the “Blend_start” to the “Blend_0p6” blend. Note how the T240 mixed blend output is ‘-7’, indicating that the weight placed on the NEXT blend output is > 0.

If you want to see the CURR and NEXT filter banks for a given sensor DOF, you can click on the CURR and NEXT buttons to bring up the standard MEDM screens:


S1ISIITMX_ITMX_ST1_BLND_CPSX1.adl
S1 - FEC    S1ISI-ITMX_ST1_BLND_CPSX1    Thu Sep 8 17
EXCMON 0.000
INMON -226.518
OFFSET 0.0
Blend_start Blend_0p6 Blend_HAMFE
1.000
LIMIT 0.0
Ramp Time (s) 0.000
HOLD OUTPUT
-189.985
OUT16 -190.269
OUTPUT -190.3
OUTMON

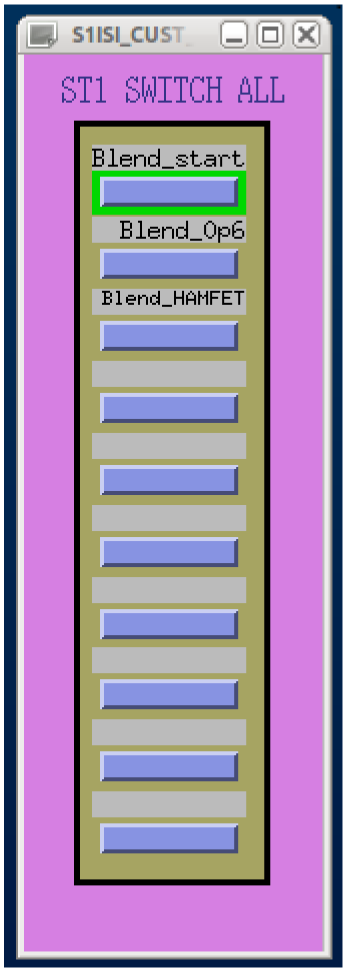
S1ISIITMX_ITMX_ST1_BLND_CPSX2.adl
S1 - FEC    S1ISI-ITMX_ST1_BLND_CPSX2    Thu Sep 8 17
EXCMON 0.000
INMON -238.413
OFFSET 0.0
Blend_start Blend_0p6 Blend_HAMFE
1.000
LIMIT 0.0
Ramp Time (s) 0.000
HOLD OUTPUT
-180.320
OUT16 -191.293
OUTPUT -191.3
OUTMON

In the above screenshot, we are switching from “Blend_0p6” to “Blend_start”.

**WARNING: Do not modify the screens above. Proper blend switching should be done at a higher level screen since the MEDM screens and Perl script assume that the user cannot directly control these screens.**

## Switching Multiple DOF

For convenience, we’ve included a “SWITCH_ALL” MEDM screen, which can be accessed from the top of the overall blend filter MEDM screen. This screen allows switching all the DOFs for a given stage to the selected blend:



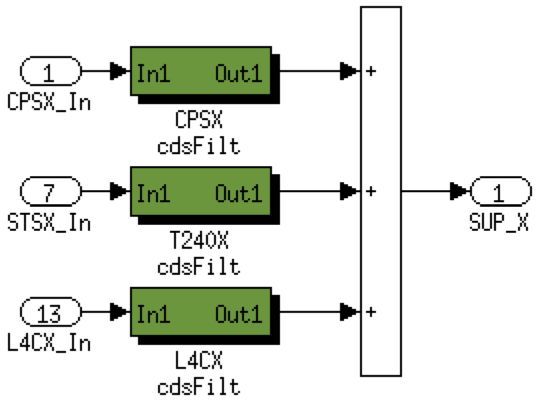
Note that to use this functionality, matching blends must be loaded into the same filter modules for ALL the degrees of freedom for a given stage. In this case, FM1 has the “Blend_start” blend loaded in for all the stage 1 DOF, FM2 has “Blend_0p6”, etc.

## Implementation:

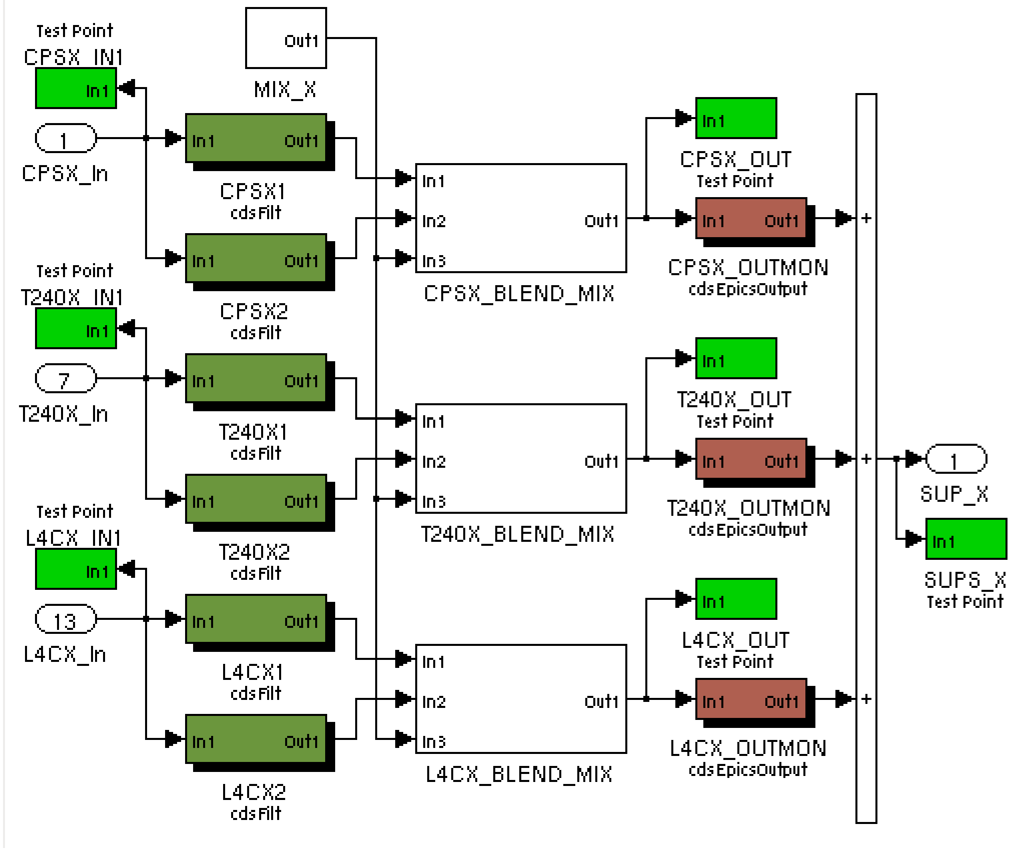
To implement real-time blend-switching, we rely on a modified Simulink diagram, C code to ramp between blends, and a Perl script to provide hierarchical control of the switching.

### Simulink

The old blend filter diagram for a single DOF is shown below:

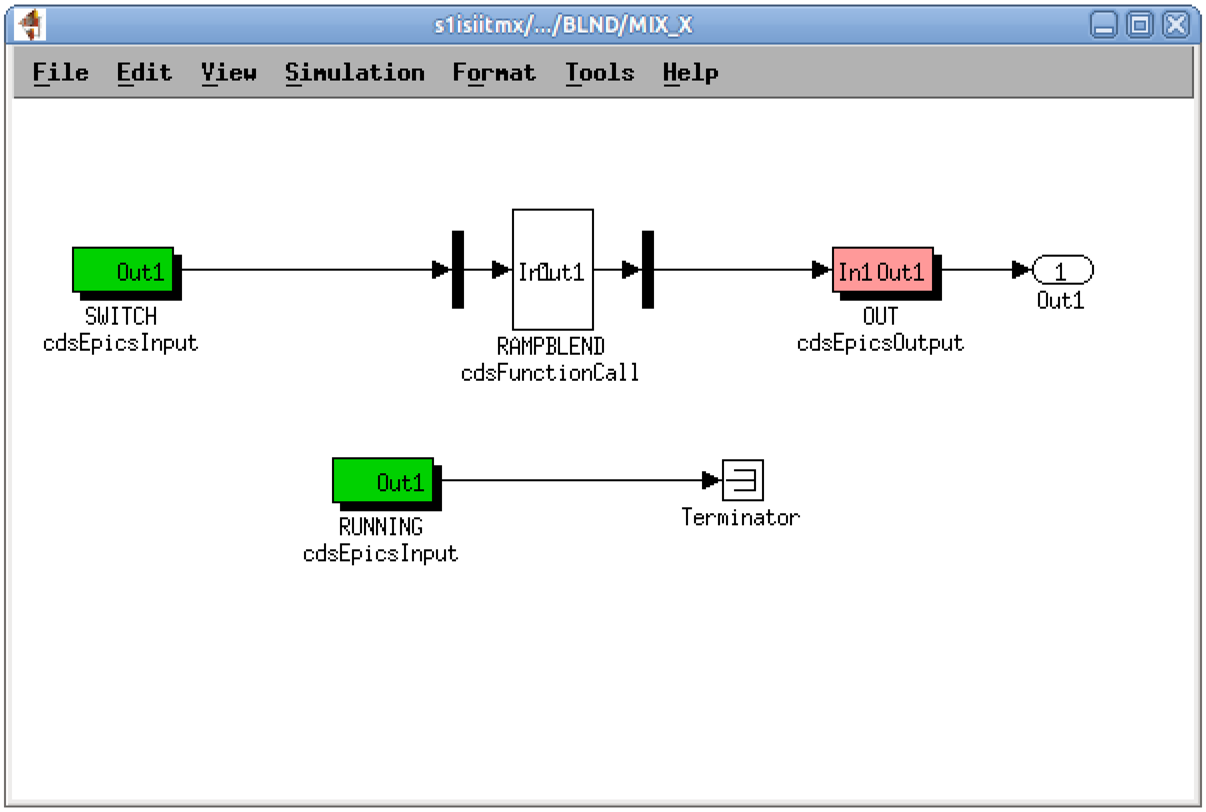


The modified diagram looks like:



From left to right, we read in the sensor inputs for a given DOF. This input is fed into the CURR and NEXT blend banks for each sensor (labeled 1 and 2, respectively). The outputs of the blend banks are mixed in the BLEND_MIX block, with the ramping variable set by the MIX_X block. The mixed output is fed into a cdsEpicsOutput block and all the sensor mixed outputs are summed to create the supersensor signal. We also include test points before and after the blending for backwards compatibility in naming conventions and writing the blend output to the framebuilder at the model rate.

The MIX_X block is shown below:



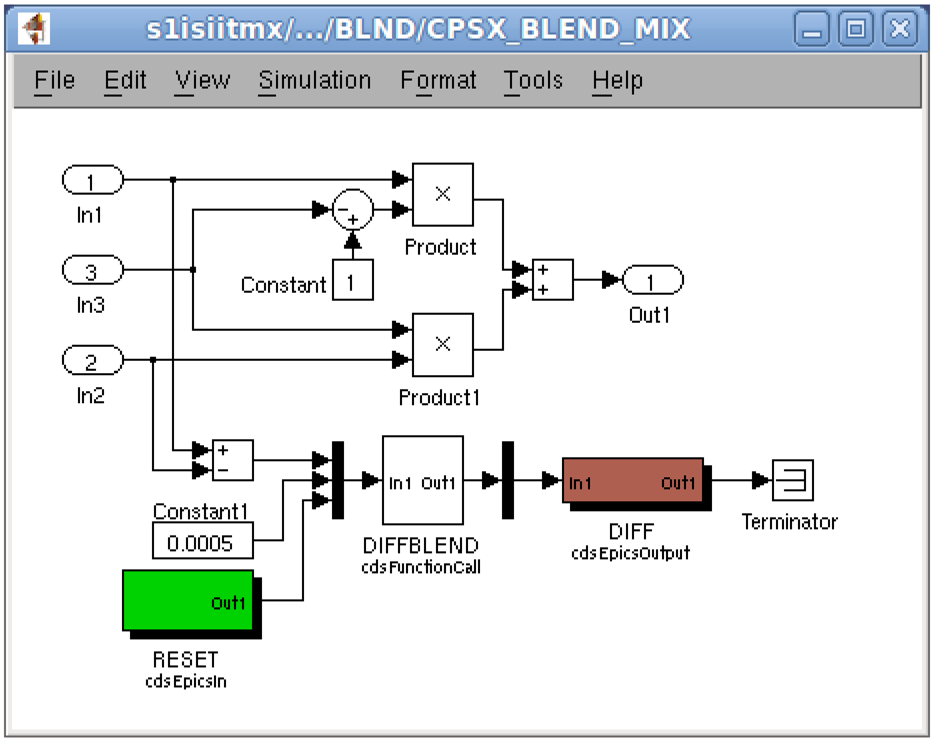
This block contains two cdsEpicsInput variables: SWITCH and RUNNING. SWITCH is set by a Perl script to either 0 or 1. Rising and falling edges on SWITCH determine when the RAMPBLEND C function will generate a ramping variable OUT > 0, which is a cdsEpicsOutput variable. RUNNING is set to 1 by the Perl script while blend switching is occurring and 0 when it has completed, which allows the Perl script to ignore additional switching requests while blend switching is occurring.

## Ramping C code

The RAMPBLEND C function is defined in the BLENDMIXER.C file below:

```
/* BLENDMIXER.c
 * Function: RAMPBLEND
 *
 * This function outputs a variable that smoothly ramps from 0 to 1 on the rising edge
 * of an input, and then ramps back down from 1 to 0 on the falling edge of that same input.
 * The 'smooth' ramp is a fifth-order polynomial that represents the minimum-jerk trajectory
 * from 0 to 1. (see http://www.shadmehrlab.org/book/minimum_jerk/minimumjerk.htm)
 *
 * On a rising edge, the increment is set to +1 and the timer is started. The timer continues
 * to be incremented on each cycle until it reaches the total_max_time allowed. Then, the
 * function waits for a falling edge from the input, decrements the timer, and increments by -1
 * until the output value goes to 0.
 *
 * Authors: CJK RK
 * September 7th 2011
 *
 */

void RAMPBLEND(double *argin, int nargin, double *argout, int nargout){
	//timer increments once per cycle
	static int timer = 0;
	static int prev_ramp_switch = 0;
	static int increment = 0;
	int cur_ramp_switch = argin[0];
	//total mix time = # seconds * 4096 cycles/sec (model rate)
	int total_mix_time = 5 * 4096;
	double mix_value = 0;
	// STATE 1: Start ramping from 0 to 1 if the timer is 0 and switch gets turned on
	if( timer == 0 && prev_ramp_switch == 0 && cur_ramp_switch == 1){
		increment = 1;
		++timer;
	}
	//STATE 2: Start ramping from 1 to 0 if timer is maxed and switch gets turned off
	else if(timer == total_mix_time && prev_ramp_switch == 1 && cur_ramp_switch == 0) {
		increment = -1;
		--timer;
	}

	// only increment the time and calculate the mix value if the timer
	// has a value between 0 and the total_mix_time
	if(timer > 0 && timer < total_mix_time){
		double timer_frac = (double) timer / total_mix_time;
		double timer_frac_cubed = timer_frac * timer_frac * timer_frac;
		mix_value = 10 * timer_frac_cubed - 15 * timer_frac_cubed * timer_frac + 6 *
timer_frac_cubed * timer_frac * timer_frac;
		timer += increment;
	} else if(timer == total_mix_time) { mix_value = 1; }

	// Set the previous switch value to the current value to prevent rapid flipping
	prev_ramp_switch = cur_ramp_switch;
	argout[0] = mix_value;
}
```

The BLEND_MIX block is shown below:



Input 1 (the CURR blend) and Input 2 (the NEXT blend) are combined using Input 3 (ramping variable X) via the following logic:

CURR * (1 – X) + NEXT * (X) = Mixed Blend Output (Out1)

During normal operation, X = 0, so the output is simply the CURR blend. During switching X smoothly varies from 0->1 (and back from 1->0), so the output is a weighted sum of the CURR and NEXT blends.

Note that we have included a cdsEpicsOutput called DIFF. The motivation for this variable is the following – rather than switching for a fixed time, we switch and wait for the DIFF signal to settle before considering our switch complete. DIFF is a low-pass filtered signal indicating the difference between the CURR and NEXT bank outputs. The low-pass filter time constant depends on the smoothing factor α (Constant1) via the following relation:

$$RC = T_s \left(\frac{1-\alpha}{\alpha}\right)$$

where $T_s$ is the model sampling period. We also include a RESET EPICS variable. When ON (set to 1), the low-pass filter clears its history and doesn't filter the signal (outputs the raw difference signal). We have chosen not to use the DIFF signal for switching because testing showed our CURR and NEXT signals converged quickly in the fixed time approach. However, the filter has been tested and could be used in future implementations.

## Low pass filter C code

The C-code to implement the low-pass filter is very straightforward:

```c
/* BLENDDIFF.c
 * Function: DIFFBLEND
 *
 * Implement a low-pass filter to smooth the difference signal between the CURR and
 * NEXT banks.  If the reset is engaged, we output the non-filtered (real-time) difference
 * and clear the filter history.
 *
 * The corner frequency of the filter depends on the coeff as follows:
 *         f = 1 / (2*pi*RC), where RC = Ts*(1 - coeff)/coeff
 * So for a 4096 Hz model rate and a coefficient of 0.0005, we expect a corner frequency:
 *         f = 1 / (2*pi*(1/4096)*(1-0.0005)/0.0005)) = 0.33 Hz
 *
 * Authors: RK BTL
 * October 17th 2011
 *
 */

void DIFFBLEND(double *argin, int nargin, double *argout, int nargout){

        static double prev_diff = 0.0;
        double cur_diff = argin[0];
        double coeff = argin[1];
        int reset = argin[2];

        // Reset the history
        if(reset) {
                prev_diff = 0.0;
                coeff = 1.0;
        }

        // Low-pass the difference signal
        double diff_output = coeff * cur_diff + (1.0 - coeff) * prev_diff;
        prev_diff = diff_output;
        argout[0] = diff_output;
}
```

## Perl

This Perl script switches between blend filters for the BSC-ISI. The script takes as arguments:
1) The stage for which the user wants to switch filters,
2) The filter module containing the filter to which the user wants to switch,
3) An optional list of the exact degrees of freedom the user wants to switch. If this list is omitted, all degrees of freedom are switched.

The script is rather self-explanatory, and much of the code is building the correct data structures and strings that enable easy switching later on. There is also some housekeeping that prevents the user from starting a switch while there is already a switch in progress or when attempting a switch to an empty filter module. The core of the code begins at line 74, where the second filter bank is switched on with the appropriate filter module. The script allows the filter to build up a history for five seconds, and then

begins the switching process. At line 80, the script puts a '1' into all mixing channels to start the mixing process.
At this point, the 'HI' state of the mix channel causes the RAMPMIXER.c code to smoothly interpolate the output of the blend filter from filter bank A to filter bank B along a minimum-jerk spline. Once the script detects that the output has fully switched to filter bank B, it puts a 0 into the mix channels to start the process in reverse. Since the filter history resets automatically whenever the filter module is turn off, there's no need to completely reset the history. See documentation on RAMPMIXER.c for details about how the spline is signaled to ramp up and down.

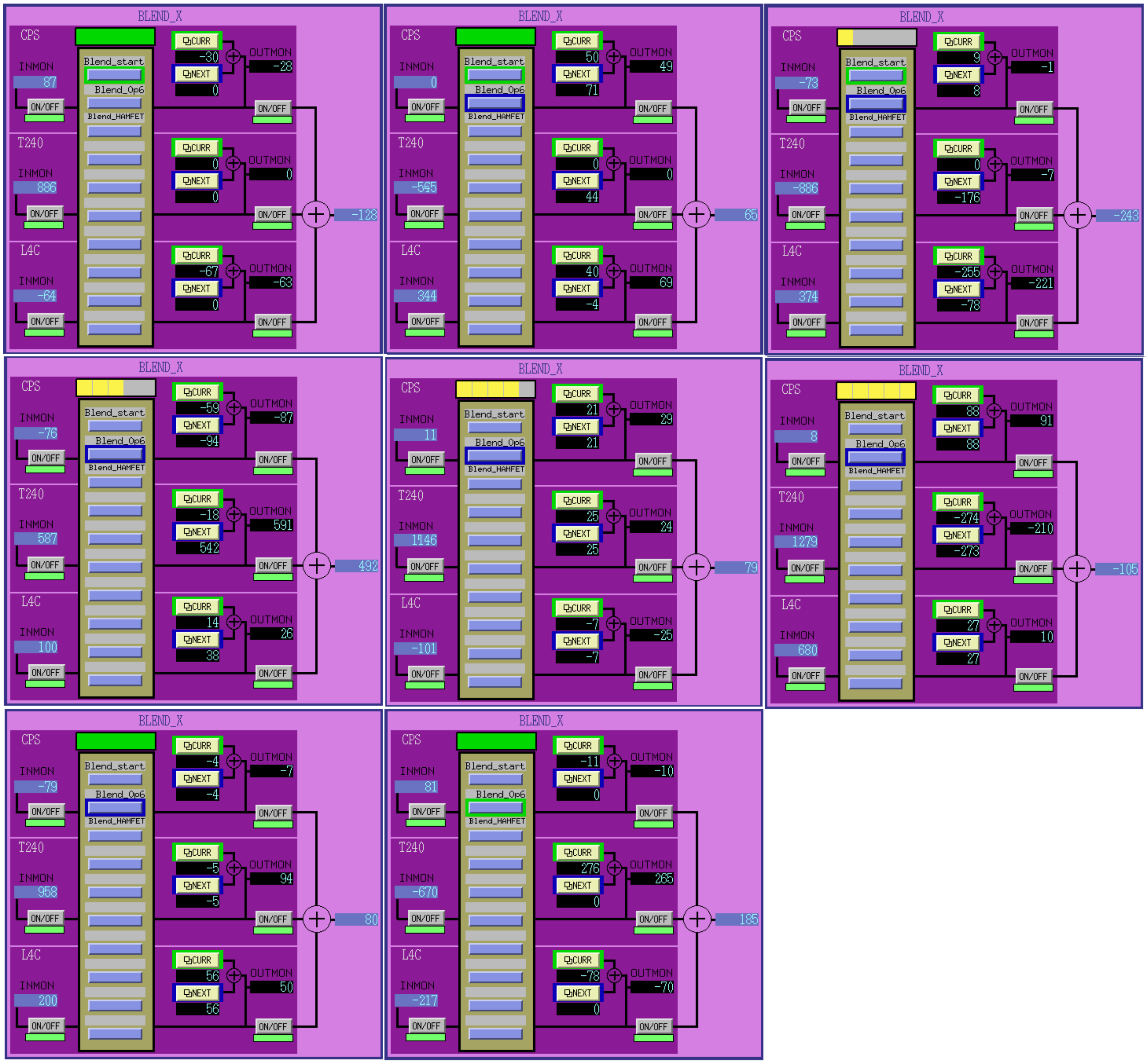
```
#!/usr/bin/perl -I /ligo/cdscfg

# switchBlendFilters
# usage: ./switchBlendFilters system stage DOF FM [args]
#
# This script switches between two different blend filters in real time.
#
# Author: CJK 2011 Sep 7
#

use strict;
use warnings;
use stdenv;
INIT_ENV(1);
use CaTools;
use 5.010;

sub usage{
        print <<USAGE
usage: ./switchBlendFilters system stage FM [dofs]

        <system> is the name of the system, formatted as ifo:sys-chamber, e.g. s1:isi-itmx
        <stage> is the stage, e.g. st1 or st2
        <FM> is the filter module to switch to, e.g. FM2
        <dof> is an optional list of the degrees of freedom to
                switch, e.g. X or RZ

USAGE
}

unless (@ARGV >= 3){
        &usage; die "Incorrect number of arguments.";
}

# Read in all of the inputs
my $SubSys = uc(shift);
eval{ caGet("${SubSys}_MASTERSWITCH") };
die "Error: bad subsys $SubSys" if $@;
my $STAGE = uc (shift);
my $fm = uc(shift);
my @DOFS = @ARGV;
die "Error: improper FM argument." unless ($fm =~ m/FM[1-9]0?/);
```

```
# Create arrays filled with the names of the switch readback channels for the correct stage
my @ST1_SENSORS = qw (ST1_BLND_CPS ST1_BLND_T240 ST1_BLND_L4C);
my @ST2_SENSORS = qw ( ST2_BLND_CPS ST2_BLND_GS13 );
if(!@DOFS) { @DOFS = qw ( X Y Z RX RY RZ ); }

my @SENSORS = ($STAGE eq 'ST1') ? @ST1_SENSORS : @ST2_SENSORS;
my $num_sensors = @SENSORS;
my $num_dofs = @DOFS;

#check to see if we're already switching for a particular dof
my @RUNNING_CHANS = map { "${SubSys}_${STAGE}_BLND_MIX_${_}_RUNNING" } @DOFS;
my @RUNNING_VARS = caGet(@RUNNING_CHANS);
foreach my $runvar (@RUNNING_VARS){
        if($runvar){
                die "Error: channel is currently switching.";
        }
}
caPut(@RUNNING_CHANS, (1) x @RUNNING_CHANS);

my @SENSOR_CHANNELS = map { "${SubSys}_" . $_ } @SENSORS;
my @SENSORDOF_CHANNELS;
foreach my $DOF (@DOFS){
        push @SENSORDOF_CHANNELS, map { $_. "${DOF}" } @SENSOR_CHANNELS;
}

# create arrays filled with names for the mixing channels and both filter banks
my @FILTERS_A = map { "${_}1" } @SENSORDOF_CHANNELS;
my @FILTERS_B = map { "${_}2" } @SENSORDOF_CHANNELS;
my @MIX_CHANS = map { "${SubSys}_${STAGE}_BLND_MIX_${_}_SWITCH"} @DOFS;
my @OUT_CHANS = map { "${SubSys}_${STAGE}_BLND_MIX_${_}_OUT"} @DOFS;

# Switch on the appropriate FM in filter bank 2
my $command = "ALL OFF INPUT OUTPUT DECIMATE $fm ON";
caSwitch(@FILTERS_B, ($command) x ($num_sensors * $num_dofs));
sleep(5);


# begin mixing
caPut(@MIX_CHANS, (1) x $num_dofs);
# wait until the output goes to 1, i.e. that we're fully operating off the new filter
(my @outputs) = caGet(@OUT_CHANS);
while($outputs[0] < 1){
        sleep(1);
        (@outputs) = caGet(@OUT_CHANS);
}

# Switch the filter bank 1 to the desired state, reset the history
caSwitch(@FILTERS_A, ($command) x ($num_sensors * $num_dofs));
sleep(5);

# begin ramping down, wait until output reaches 0
caPut(@MIX_CHANS, (0) x $num_dofs);
@outputs = caGet(@OUT_CHANS);
```

```
while($outputs[0] > 0){
        sleep(1);
        (@outputs) = caGet(@OUT_CHANS);
}

# now that we're all done, turn the second filter bank off
caSwitch(@FILTERS_B, ("ALL OFF") x ($num_sensors * $num_dofs));
caPut(@RUNNING_CHANS, (0) x @RUNNING_CHANS);
exit 0;
```

## MEDM

The following is the sequence of MEDM screens that appears while blend switching takes place (from left to right, then down to the next row):

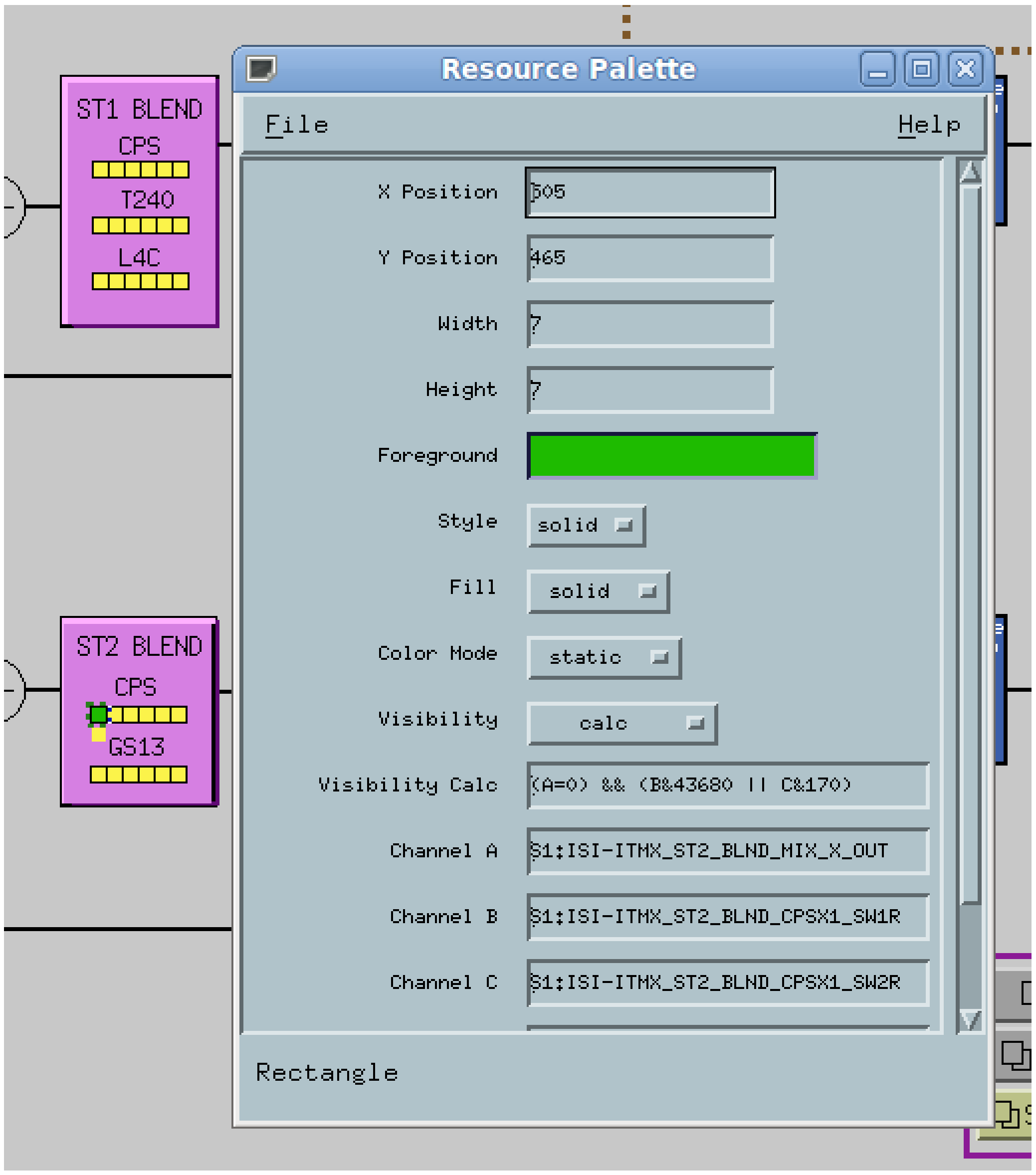


1. “Blend_start” currently active.
2. “Blend_0p6” is selected to be active in the NEXT bank, but the switching has not begun.
3. The switching has started. One yellow indicator bar means that the ramping variable > 0.

4. We have switched over to the NEXT bank. Three yellow indicator bars means the ramping variable = 1 or we are ramping down. Note that two yellow indicator bars means the ramping variable is > 0.6 or we are ramping down. The CURR bank is now on with the "Blend_0p6" selected and we will begin ramping back down (transitioning to the CURR bank). We can't see the green CURR highlight because it is covered by the blue NEXT highlight.
5. The ramping variable is <0.6 and we are ramping down.
6. The ramping variable is < 0.3 and we are ramping down.
7. The ramping variable = 0. The NEXT bank is still on so the blend is highlighted blue.
8. We've turned off the NEXT bank, so the CURR bank highlight is now visible. We're done.

## Low-level Implementation

At the highest level, we display the status of the blends in the overview screen, seen below.



X Position 505
Y Position 465
Width 7
Height 7
Foreground
Style solid
Fill solid
Color Mode static
Visibility calc
Visibility Calc (A=0) && (B&43680 || C&170)
Channel A $1:ISI-ITMX_ST2_BLND_MIX_X_OUT
Channel B $1:ISI-ITMX_ST2_BLND_CPSX1_SW1R
Channel C $1:ISI-ITMX_ST2_BLND_CPSX1_SW2R
Rectangle

Each sensor DOF is represented by a square block. By default, the blocks are red. When switching (i.e. the ramping variable is > 0), they are yellow. They are only green when the following conditions are satisfied:

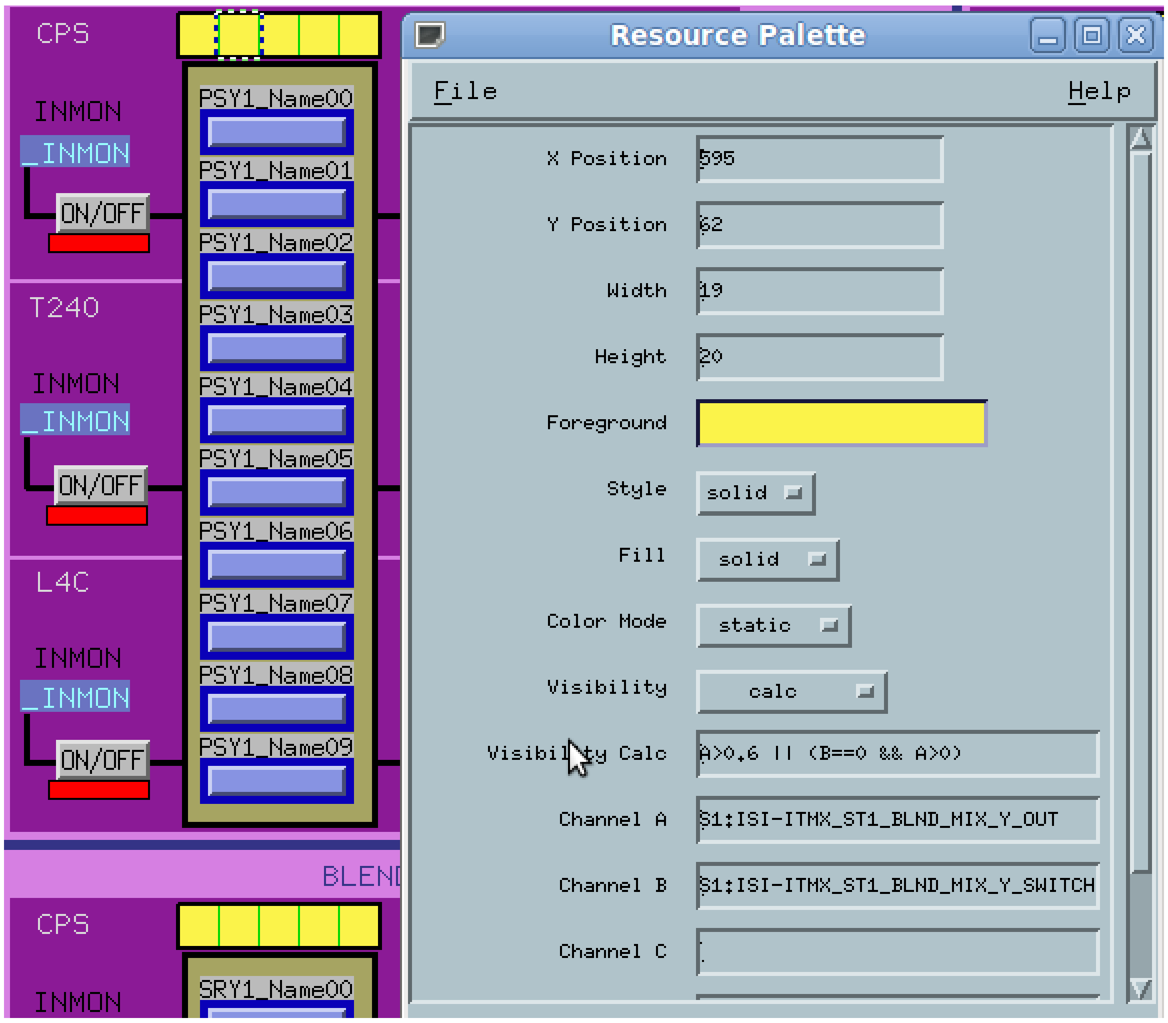
1. Ramping variable = 0
2. One of the filter modules in the CURR bank is ON

The state of the readback channels for CDS filter blocks is given by bit states in SW1R and SW2R summarized below, assuming bit 0 = 1st bit. We can AND SW1R and SW2R with the proper bitmasks to check if a FM is on.

| **SW1R** | | **SW2R** | |
|---|---|---|---|
| **Bit** | **State** | **Bit** | **State** |
| 2 | Input switch | 1 | FM7 |
| 5 | FM1 | 3 | FM8 |
| 7 | FM2 | 5 | FM9 |
| 9 | FM3 | 7 | FM10 |
| 11 | FM4 | 10 | Output switch |
| 13 | FM5 | | |
| 15 | FM6 | | |

Note that we don't check that the input and output switches for the CURR bank are ON (since we don't expect the user to modify these anyway), though one of these could be checked for as well. We cannot check for both since MEDM only allows 4 channels for visibility calculations, and we're already using 3 to check for the two conditions above.

At the blend MEDM screen level, we have the progress bar, seen below:



| | |
|---|---|
| X Position | 595 |
| Y Position | 62 |
| Width | 19 |
| Height | 20 |
| Foreground | 🟨 |
| Style | solid |
| Fill | solid |
| Color Mode | static |
| Visibility | calc |
| Visibility Calc | A>0.6 \|\| (B==0 && A>0) |
| Channel A | S1:ISI-ITMX_ST1_BLND_MIX_Y_OUT |
| Channel B | S1:ISI-ITMX_ST1_BLND_MIX_Y_SWITCH |
| Channel C | |

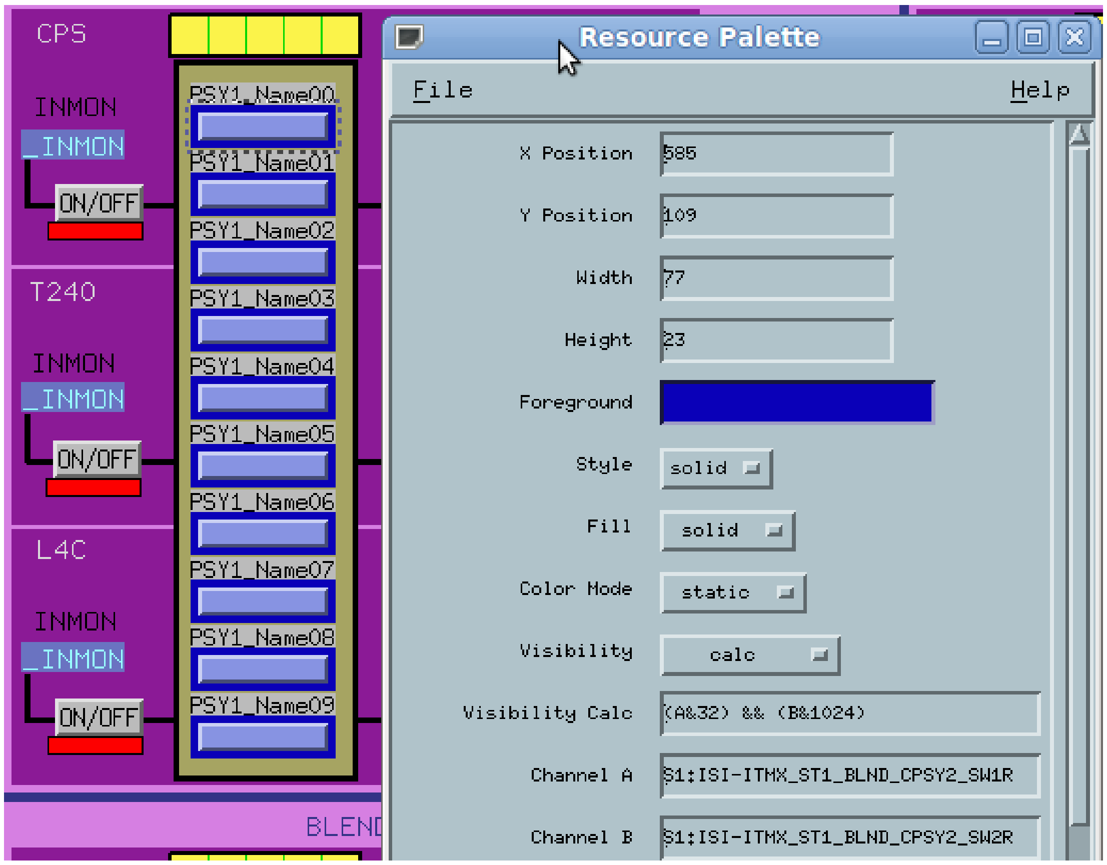
To display the progress bar, we must keep track of two EPICS channels – the ramping variable (A) and the switch (B). The tricky part is we want the progress squares to turn on when the ramping variable exceeds a certain threshold, but to stay on as the ramping variable comes back down to 0 – the switch allows this ability. By default, the switch is at 0. It flips to a 1 to start the ramp, and flips back to a 0 once the ramping variable = 1. Therefore,we use the following conditions to control when the progress squares turn on (and stay on):

| Square | Condition |
|---|---|
| 1 | A>0 |
| 2 | A>0.6 \|\| (B==0 && A>0) |
| 3 | A==1 \|\| (B==0 && A>0) |
| 4 | B==0 && A< 0.6 && A>0 |
| 5 | B==0 && A<0.3 && A>0 |

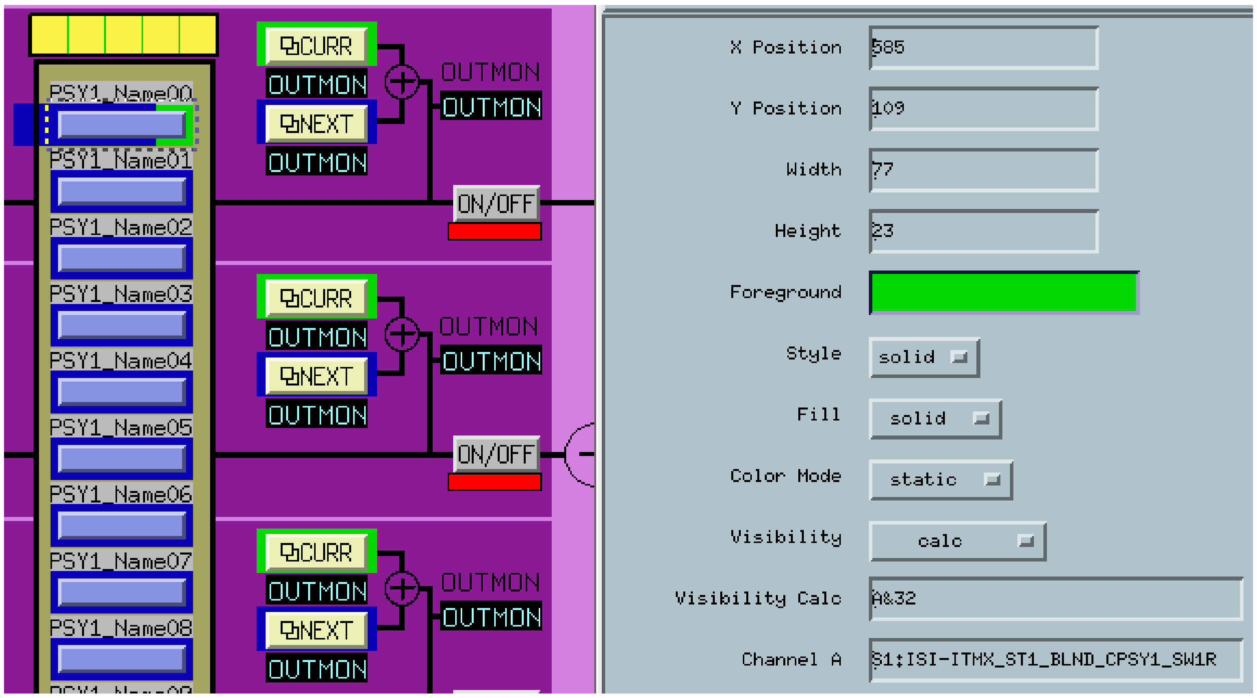
The green progress bar is on under the same conditions as the green square on the overview screen:

1. Ramping variable = 0
2. One of the filter modules in the CURR bank is ON

To indicate the CURR and NEXT blend, we highlight the CURR blend green, and the NEXT blend blue, seen below:



| | |
|---|---|
| X Position | 585 |
| Y Position | 109 |
| Width | 77 |
| Height | 23 |
| Foreground | 🟦 |
| Style | solid |
| Fill | solid |
| Color Mode | static |
| Visibility | calc |
| Visibility Calc | (A&32) && (B&1024) |
| Channel A | S1:ISI-ITMX_ST1_BLND_CPSY2_SW1R |
| Channel B | S1:ISI-ITMX_ST1_BLND_CPSY2_SW2R |



| | |
|---|---|
| X Position | 585 |
| Y Position | 109 |
| Width | 77 |
| Height | 23 |
| Foreground | 🟩 |
| Style | solid |
| Fill | solid |
| Color Mode | static |
| Visibility | calc |
| Visibility Calc | A&32 |
| Channel A | S1:ISI-ITMX_ST1_BLND_CPSY1_SW1R |

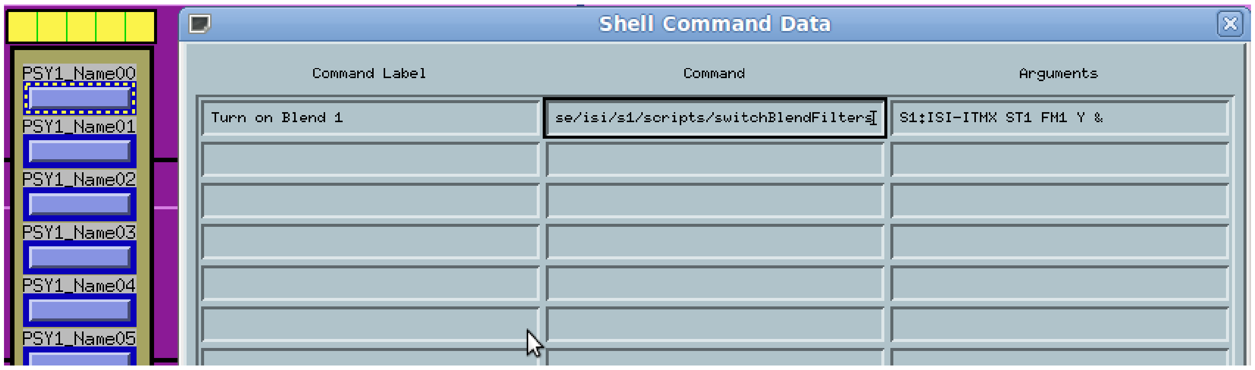
In the example above, we consider the first blend, which is loaded into FM0 in both the CURR and NEXT blend banks. We know that the state of the FM0 readback channel is available via bit 5 of SW1R (see above), so we highlight this blend green if this bit is active in the CURR blend bank.

We highlight the FM1 blend blue when the following conditions are met:

1. FM1 is turned on in the NEXT bank (bit 5 of SW1R)
2. The output of the NEXT bank is turned on (bit 10 of SW2R)

Note that we could also check that the input of the NEXT bank is switched on, but since the Perl script should automatically turn on the input and output, this would be redundant.

To interface the MEDM screen with the Perl script that controls blend switching, we use buttons that call the Perl script with the proper arguments, seen below:



| Command Label | Command | Arguments |
|---|---|---|
| Turn on Blend 1 | se/isi/s1/scripts/switchBlendFilters | S1:ISI-ITMX ST1 FM1 Y & |
| | | |
| | | |
| | | |
| | | |
| | | |

In the above example, clicking the top blend button will call the switchBlendFilters script on ITMX, with the Stage 1 Y DOF asked to switch to the blend in FM1.