TECHNICAL INSIGHTS

Custom Form Factors in the Introspect ESP Software

INTROSPECT.CA

Table of Contents

Introduction

This document will explain how to define custom form factors in the Introspect ESP Software. Defining a custom form factor allows you to do the following:

- Have a custom name for a form factor
- Specify which hardware boxes will be used
- Treat several hardware boxes as one (with channels extending across the boxes)
- Handle several hardware boxes in one Test (accessing one at a time)
- Add an "AUX box" to the mix for additional specialized functionality

Main Concepts

CUSTOM NAME

Each custom form factor has a unique name. You might want to define a custom form factor just so you can have a different name for a form factor that you use frequently. For example, if you use the SV1C_8C12G form factor for cable testing, you might want to have a custom form factor that is identical to the SV1C_8C12G but is named "CableTester". The names of custom form factors appear in the menu of available form factors at application startup, and having a custom name for a form factor that you don't otherwise customize might make it easier to pick out from that menu.

HARDWARE BOXES TO CONNECT TO

If there are multiple hardware boxes connected to your computer (via USB), the Introspect ESP Software decides which box to connect to by looking at the FTDI serial numbers of the available boxes. Different types of Introspect hardware usually have different patterns of FTDI serial numbers, and so the software can choose the appropriate box to connect to when the boxes are of different types. But if you have more than one box of the same hardware type connected to your computer, the software will randomly choose between these boxes when deciding which one to connect to.

Defining a custom form factor provides an easy way to specify which hardware box you want to connect to. You can specify that this custom form factor should only connect to a box with precisely specified FTDI serial numbers, or you can specify a pattern of FTDI serial numbers that should be looked for. Hence, you might want to define a custom form factor just so you can tie it to particular hardware boxes.

TREATING SEVERAL HARDWARE BOXES AS ONE VIRTUAL BOX

If you have more than one hardware box of the same type, you can define a custom form factor that will make those multiple boxes appear as one virtual box. This is referred to as "horizontal stacking" since it is as if the boxes are laid out in a horizontal row and combined together. If you have N boxes, each with K channels, the custom form factor will have (N * K) channels. The channel numbers will range from 1 to (N * K) and you can use any subset of these channels in the components that address channels (e.g. the RxChannelList component). This allows you to have one Test Procedure that controls all of the N boxes. The channels will not be precisely synchronized – for example, if you use the TxChannelList component to start patterns on all channels, the patterns will start on the first box a few milliseconds before those on the second box. Some form factors provide additional mechanisms to synchronize the boxes.

When you specify what FTDI serial numbers to be used for each of the boxes in a custom form factor, that also defines the order of the boxes for the purpose of channel assignment. For example, if you have two SV5C_16C12G boxes, "box1" will have channels 1-16, and "box2" will have channels 17-32.

No new software licenses are needed to use custom form factors. If you have a license for use of each of the boxes separately, that will allow use of a custom form factor that virtually bonds several boxes into one.

HANDLE SEVERAL HARDWARE BOXES IN ONE TEST

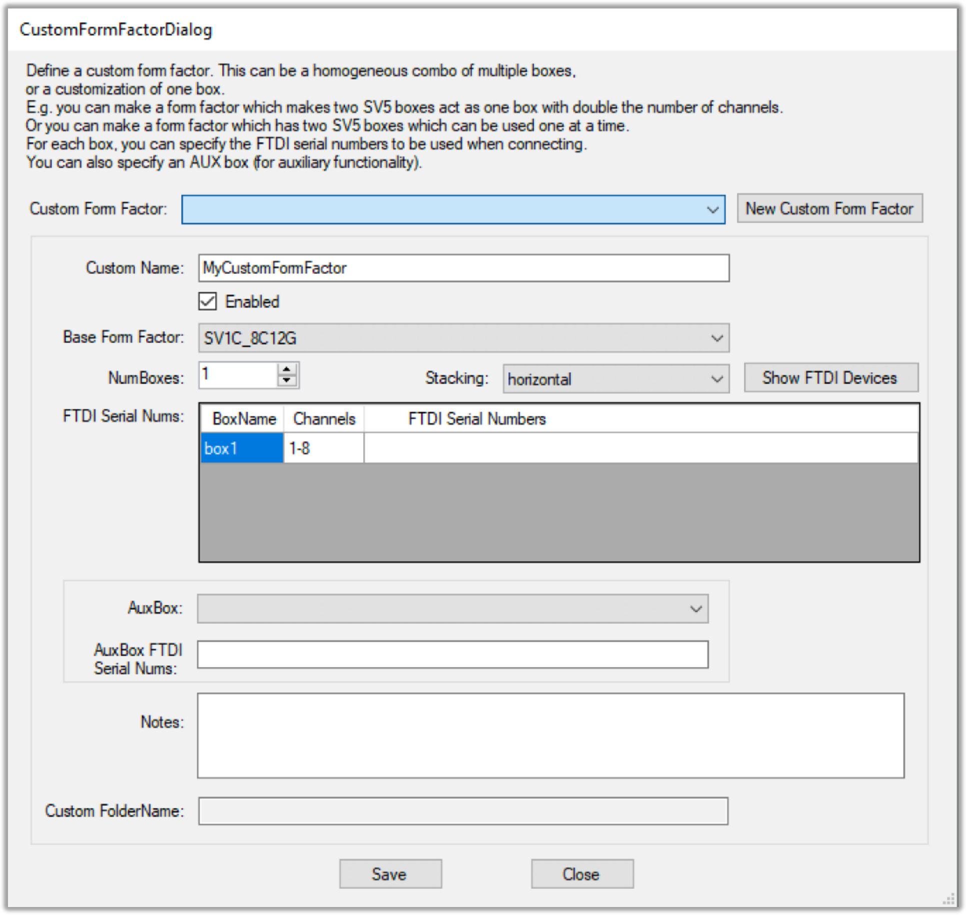
If you have more than one hardware box of the same type, an alternative to the "horizontal stacking" described in the previous section is "vertical stacking". A custom form factor in "vertical stacking" mode allows you to control multiple hardware boxes in one Test, but only addressing one box at a time. You can think of the boxes as being stacked in a vertical pile with the boxes shadowing each other. Each box will have its usual K channels and you can do operations involving the K channels of the "current" box. E.g. if you have two SV4E_4L2G_MIPI_DPHY_CAPTURE boxes, "box1" will have lanes 1-4 as usual, and "box2" will have lanes 1-4 as usual. You can then use this custom form factor to read from two different MIPI TX DUTs (one at a time). You can specify your own names for the boxes to make it easier to keep track of which is which.

AUX BOXES

Some Introspect hardware can be used in conjunction with other boxes in a custom form factor to provide additional specialized capabilities. For example, the SV3C_VECTORBLASTER form factor provides low-level methods for sending ATE-style "vectors". A custom form factor can include an AUX box to make its capabilities available in the software. AUX boxes don't have software components – they provide only low-level commands.

Defining a Custom Form Factor in the GUI

You bring up the Custom Form Factors dialog by choosing the menu item "Custom Form Factors" in the "Tools" menu of your Test window. The Custom Form Factors dialog looks like this:


CustomFormFactorDialog

Define a custom form factor. This can be a homogeneous combo of multiple boxes,
or a customization of one box.
E.g. you can make a form factor which makes two SV5 boxes act as one box with double the number of channels.
Or you can make a form factor which has two SV5 boxes which can be used one at a time.
For each box, you can specify the FTDI serial numbers to be used when connecting.
You can also specify an AUX box (for auxiliary functionality).

Custom Form Factor:

Custom Name: MyCustomFormFactor

Enabled: ✓

Base Form Factor: SV1C_8C12G

NumBoxes: 1

Stacking: horizontal

FTDI Serial Nums:

BoxName	Channels	FTDI Serial Numbers
box1	1-8	

AuxBox:

AuxBox FTDI Serial Nums:

Notes:

Custom FolderName:

The "**Custom Form Factor**" drop-down menu at the top shows the names of existing custom form factors. Here we haven't yet defined any custom form factors, so that menu is empty.

The "**Custom Name**" field is where you enter the name that you want for your custom form factor. The name must be a valid Python identifier, so it can't have spaces in it, and it must start with a letter or an underscore.

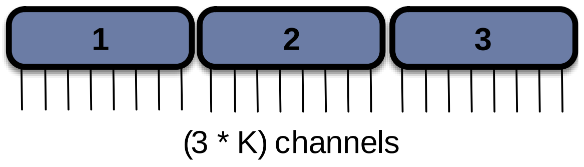
The "**Enabled**" checkbox is usually left checked. If you wanted to disable a custom form factor so it wouldn't appear in the form factor menu at application startup, you could uncheck this checkbox.

The "**Base Form Factor**" drop-down menu is where you specify the existing form factor that your custom form factor is based on. Here it is showing SV1C_8C12G since we started the application using an SV1C_8C12G form factor.

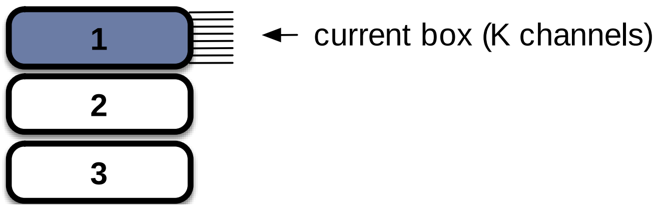
The "**NumBoxes**" control is where you specify how many boxes (of the type used with "Base Form Factor") you want to include in your custom form factor. You will need to have that number of hardware boxes of that type when you use the custom form factor. If you are just defining a custom form factor in order to get a custom name, or to specify the FTDI serial numbers to avoid ambiguities at connection, leave "NumBoxes" at 1.

The "**Stacking**" drop-down menu allows you to specify whether to use "horizontal" or "vertical" stacking. As mentioned above, "horizontal" stacking gives a custom form factor that makes the N boxes (each with K channels) act like one virtual box with (N * K) channels, while "vertical" stacking gives a custom form factor where the N boxes shadow each other and only one box (the "current" box) is available at a time.

Horizontal stacking:



Vertical stacking:



The "**FTDI Serial Nums**" area is where you specify the FTDI serial numbers for each hardware box to be used with your custom form factor. You can specify the full FTDI serial number, or you can give a regex (regular expression) that specifies the pattern that the serial number has to match. If you don't specify an

FTDI serial number for a box, the default pattern for that form factor will be used. The button "Show FTDI Devices" will popup a dialog showing the serial numbers of the FTDI devices currently on the USB bus. This is often helpful in filling in the FTDI serial numbers you want to be used with your custom form factor.

Note that some form factors have more than one internal "subpart" and for these you should specify the FTDI serial number for each subpart (in a comma-separated list). Usually the first subpart's FTDI serial number ends with "A" and the second subpart's FTDI serial number ends with "B".

You can also specify your own names for the boxes in the "FTDI Serial Nums" area. The names of boxes need to be valid Python identifiers, so no spaces are allowed.

The "Channels" column of the "FTDI Serial Nums" area is read-only. It shows how the channel numbers will be assigned to the boxes – this depends on whether the Stacking is "horizontal" or "vertical".

The "**AuxBox**" drop-down menu is where you specify the form factor that you want to use as an "AUX box". Usually you don't want an AUX box and so you leave this as blank (or choose "(None)" from the menu).

The "**AuxBox FTDI Serial Nums**" area is where you specify the FTDI serial numbers to be used with the AUX box.

The "**Notes**" area is where you can add your own notes about your custom form factor. These notes are just for your benefit – the software doesn't do anything with them.

The "**Custom FolderName**" field shows the name of the folder where the config file for this custom form factor is stored. This is a read-only field. It is blank until the custom form factor has been saved.

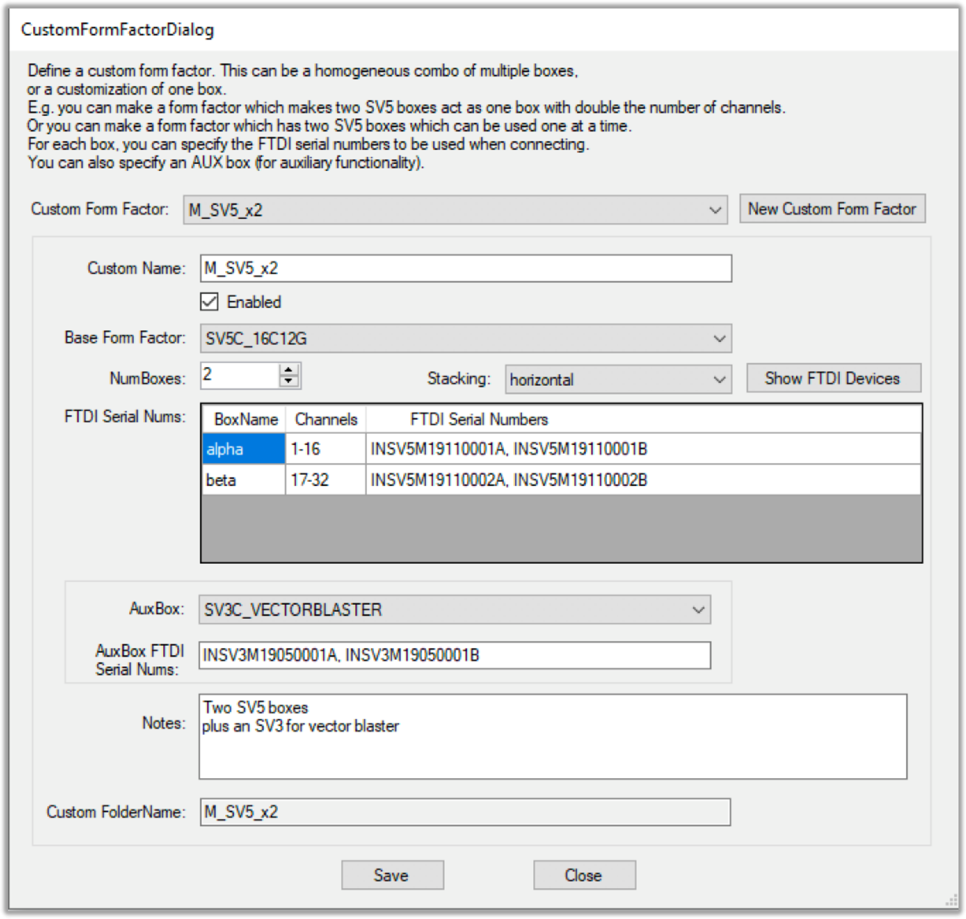
The "**Save**" button saves the currently displayed custom form factor configuration to a sub-folder under "<MyDocuments>/Introspect/Config/CustomFormFactors".

After you save your newly defined (or edited) custom form factor configuration, you can proceed to define another custom form factor by clicking on the "New Custom Form Factor" button. If you are finished with defining custom form factors, click the "Close" button to close the Custom Form Factors dialog.

After making changes with the Custom Form Factors dialog, you will need to quit the GUI and re-launch it in order to use the new custom form factor (or to have editing changes take effect).

Note: It is also possible to define a custom form factor in a **script** by using the SVT-level function `'defineCustomFormFactor'`.

If you defined a custom form factor with two SV5 boxes and an AUX box (for vector blaster functionality), this is what the Custom Form Factors dialog would look like after you clicked "Save":



CustomFormFactorDialog

Define a custom form factor. This can be a homogeneous combo of multiple boxes,
or a customization of one box.
E.g. you can make a form factor which makes two SV5 boxes act as one box with double the number of channels.
Or you can make a form factor which has two SV5 boxes which can be used one at a time.
For each box, you can specify the FTDI serial numbers to be used when connecting.
You can also specify an AUX box (for auxiliary functionality).

Custom Form Factor: M_SV5_x2

Custom Name: M_SV5_x2

☑ Enabled

Base Form Factor: SV5C_16C12G

NumBoxes: 2

Stacking: horizontal

FTDI Serial Nums:

BoxName	Channels	FTDI Serial Numbers
alpha	1-16	INSV5M19110001A, INSV5M19110001B
beta	17-32	INSV5M19110002A, INSV5M19110002B

AuxBox: SV3C_VECTORBLASTER

AuxBox FTDI Serial Nums: INSV3M19050001A, INSV3M19050001B

Notes: Two SV5 boxes
plus an SV3 for vector blaster

Custom FolderName: M_SV5_x2

Sharing a Custom Form Factor Definition

The custom form factor configurations are stored in CSV files in sub-folders under "<MyDocuments>/Introspect/Config/CustomFormFactors". You can share a custom form factor configuration with others by copying the appropriate sub-folder onto their computers. But be careful that the 'customName' (in the "config.csv" file) doesn't collide with the custom name of one of their existing custom form factors.

Using a Custom Form Factor

To use a previously defined custom form factor, you choose its name from the menu of form factors at application startup. Note that the "Filter" field is often useful since you can type part of the custom form factor in that field so that the menu will only show form factors matching that partial name.

You can then use the Introspect ESP Software components as usual (but with more channels available if your custom form factor has more than one box and has "horizontal" stacking). For example, if your custom form factor was defined to include two SV5C_16C12G boxes in "horizontal" stacking mode, you could specify the channels of an RxChannelList component as 1-32 and use it to do a BertScan over all 32 channels (2 boxes) with one command. The allocation of channels is as discussed above: channels 1-K for the first box, channels (K+1)-2K for the second box, etc.

If you need to use low-level IESP methods, you can obtain an instance of the IESP like this:

```
    iesp = getIespInstance()
```

The 'iesp' variable thus obtained refers to the custom form factor and can be used to send commands to the channels of each hardware box – the allocation of channels is as discussed above.

CONNECTING TO HARDWARE

If your custom form factor has multiple boxes in "**horizontal**" stacking mode, when you connect to the hardware using the GUI, all of the boxes will be connected and hence all the (N * K) channels will be available for use.

If your custom form factor has multiple boxes in "**vertical**" stacking mode, when you connect to the hardware using the GUI, all of the boxes will be connected. And the first box will be the "current" box, so only that one box will be available for use immediately after connection. Components can be defined with the usual number of channels (or lanes) and component operations will be directed to the "current" box. You can switch to another of the boxes using the SVT-level function 'setCurrentBox'. For example:

```
setCurrentBox('alpha')
```

You can find out which box of a vertically-stacked custom form factor is the current one using the SVT-level function 'getCurrentBox'.

If you want to do the connection programmatically (instead of having the GUI automatically connect when you press the "Run" button), you can suppress the auto-connection by adding the following as the first line in your Test Procedure:

```
#! DISABLE AUTOCONNECT
```

And then you can call the IESP-level 'connectToHardware' method like this:

```
iesp.connectToHardware()
```

Calling 'connectToHardware' with no arguments like above will connect to all of the boxes of your custom form factor.

CLOCKING

If your custom form factor has multiple boxes in "**horizontal**" stacking mode, the usual GlobalClockConfig component will have been replaced by a instance of MultiBoxClockConfig which allows you to specify the clock parameters (dataRate and reference clock) on a per-box basis. The default for MultiBoxClockConfig is "daisy-chained" reference clocking where the first box has an internal reference clock and an active outputClockA, and the remaining boxes have external reference clocks with frequency equal to that of the first box's outputClockA. Of course, to use this default setup you need to wire the boxes appropriately, with the output reference clock A of the first box connected to the input reference clock of the second box, etc.

If your custom form factor has multiple boxes in "**vertical**" stacking mode, you specify the clocking parameters as usual. When the ClockConfig component 'setup' method is called, it will apply to the "current" box. If you want to have different clocking on each box, you can create a separate ClockConfig component for use with each box and call the 'setup' method on the appropriate one after switching to a box with 'setCurrentBox'.

ISSUING AUX BOX COMMANDS

If your custom form factor has an AUX box, you need to first get an Python variable representing an instance of the AUX box like this:

```
auxBox = iesp.getAuxBox()
```

Then you can invoke the low-level methods of the AUX box (what would be IESP-level methods if you were using the AUX box form factor standalone) by using 'auxBox' instead of 'iesp'. For example, with the SV3C_VECTORBLASTER, you could do:

```
auxBox.sendVectorsFromFile(vectorFilePath)
```

Revision Number	History	Date
1.0	Document Release	July 9, 2020

INTROSPECT.CA