

Tutorial for Developing VMI 2.0 Devices

This tutorial provides step by step procedure for developing VMI devices. Developing VMI devices is essentially easy once you have understood the functions for handling irbs and slabs.

The first step to develop device is to figure out what type of device you need to develop. There are essentially two types of devices:

1. Filter devices: These are devices that are part of the chain and are not the source or sink of data. These just change the data that they get in the irb and pass the irb to the next device in the chain. Best understood by an example, compression device is a filter device for it takes the data it receives in the irb, compresses it (i.e changes the data), and sends the modified data to the device below it in the chain. The simplest example of the filter device is a pass through device, which essential takes the irb containing data and passes the data unmodified to next device in the chain.
2. Source or sink device: These are devices those are source or sinks of data in the chain. For example the TCP device in the send chain is the device that takes the data in the irb and puts it into the network and hence it is an example of a sink device as it is a sink of data. It might complete the irb or it may pend it. An example of the source device is the TCP device at the receive chain it takes the data off the network and generates an irb for it, hence it is a source of data. Thus source/ sink devices are different from filter devices as they are source or sinks of data and they complete or pend or generate irbs while filter devices are those that just change the data in the irbs and pass the irb along to the next device.

Developing filter devices for VMI 2.0

The following is the step by step procedure for developing filter devices for VMI 2.0. The passthru device provided should be treated as a guide for it implements the basic functions required by any filter device.

Change the Spec file

Add entry for your device in the specification file. Look at the sample specification file provided. You need to add information about your device as shown in the specification file. Indicate the GUID (any string which uniquely specifies your device), name, path where the shared object can be found so that it can be loaded from there, a list of args as `<arg num="1">`, `<arg num="2">` , so on.... these args provided to your device initialization functions as input arguments. Then specify the name of your initialization function as `<initfn>`, also the termination and startIO function. Each of these 3 functions is essential for the device. Each of them is described below. You can just implement these 3 functions and you have a working device, isn't that easy???

Initialization function

1. Initialization function: (In passthru device it is called: PassThruInit) The input argument (args) as shown in the passthru device is the list of arguments specified in the device section of the specification file. The other input argument is the void pointer to the device. This pointer is the pointer to your device structures and is a opaque pointer, you should store this pointer in a global variable as this will be required by the other functions of the device.

So do this in initialization function. Store the device pointer in the global variable as shown in passthru device. Also initialize any data structures that you might need for your device. For example in compression framework initialize functions of the compression devices are called in the initialize function of the framework. Return VMI Success as VMI_STATUS if everything was successful.

2. Termination function: Any clean up operations in this function. Return success as status if everything is successful or return error.

3. StartIO function: Here is where you do all the processing (filtering) of data in the irb. Here is where you make or break your device. What you have done till now is cake walk and this function is easy to implement in basic format that is in the way it is implemented in the passthru device, but may be challenging otherwise. But you love challenges right??

You are passed the irb and a context as arguments. What is an irb? In simplest form it is a data unit in VMI, you are getting data inside the irb. It also contains zillions of other information but due to excellent abstraction you need not worry yourself about it. So you got the data that the device above you (lower in rank) in the chain sent to you. Your job modify the data the way you want and send the modified irb to the next device. Sounds trivial right, well it is, if you just hang in there and do it. Implement this function step by step as shown here.

The other argument is a context pointer, it is a void pointer. What is a context? Simple it is block of memory, what it contains? Anything you want. Now you are writing a device and you want to store some information that will help you process irbs, for the irbs in this chain. So information specific to your device for that particular chain your device is on. What information? Best explained with an example, like in compression framework you want to know which all compression algorithms are supported for a particular chain. This is information which is useful for each irb that comes on that chain right? You need to know which all compression algorithms are on the chain so that you can compress the data using those and ONLY those algorithms so why not put this information you put in a data structure and pass this information as a context pointer. So that every irb that you get automatically comes with this context pointer. How do you get this in every irb? You need not worry about it. The core takes care of this. All you need to do is give a pointer to a data structure that contains information that you will require for each processing of irb. So with each and every irb you get this information. So now you ask how do i assign this information to the context pointer in the first place. Okay i will tell you this in just a minute.

Back to startIO function.

1. All irbs have a stack location for each device in the chain. These stack locations contain the data. But it contains millions of other information vital for your device. So now every device has a stack location. Everything that is essential for your device in the current stack location, you get a pointer to that using the macro `VMI_IOMGR_IRB_CURRENT_STACK`, store the pointer in `PVMI_STACK` variable. Remember you are a filter device, your purpose is to take the data in this irb modify it and pass the modified data to the next device, the next device would also get its data from the stack. So the next device would get its data from the next stack location, this stack location would be the current stack location for the next device. But for your device it is the next stack location. So your purpose as a filter device, is to get the data from the current stack location modify it and store it in the next stack location so that the next device can retrieve it. Having said use the `VMI_IOMGR_IRB_NEXT_STACK` marco to retrieve the next stack from the irb and store it in another `PVMI_STACK` pointer. So you got it get the data from the current stack pointer modify it and store it in the next stack pointer location and you are done. Sounds easy?? It is trust me.
2. All irbs have a field called the `COMMAND` field, which defines the irb. You can't process the irb until you know what type of irb you got right? For example if you get the Attach irb that is the first irb your device gets to attach itself to the chain. Your device need to attach itself to the chain right, so that it gets the subsequent data irbs. So your device needs to know what kind of irb it has got so that it can process the irb according. So get the command using the macro `VMI_IOMGR_GET_COMMAND` from the current stack location, as i said all your device data is in the stack including the command so get your command from the stack and see what type of irb you got. Best way to do that is use a switch case statement as shown in the pass thru device to do actions based on the name of the command.

The fun starts now!!! Don't look so scared. It is fairly easy.

TYPES OF COMMAND AND CORRESPONDING ACTION YOUR DEVICE NEEDS TO TAKE

The return value of `startIO` function is `VMI_STATUS` which is whether success or error. Also the irb needs to be marked complete by using the macro `VMI_IOMGR_MARK_IRB_COMPLETE` before returning from this function.

1. `VMI_IRB_ATTACH`: When the device is attached to the chain, this is the first irb your device gets. A device can be attached to multiple chains. For example the TCP device is attached to the send chain and the receive chain. For each instance of the device one can provide as stated above some arguments in the chain section of the specification file. These arguments are essential used during the attach process. So the arguments provided for the device in the chain defination section of the spec file can be obtained from the stack using `VMI_IOMGR_GET_INPUT`. Apart from using these arguments to do any initialing that needs to be done before the device can start receiving data irbs. One also needs to set the context

- (remember the context for the device described above) so that all subsequent irbs received by this device on this chain get this information stored in the form of context. Using the macro `VMI_IOMGR_SET_OUTPUT`, as shown in `passthru` it is set to `NULL` can set the context, since `passthru` device does not want to store any information it needs for subsequent irbs. Therefore the purpose of `attach irb` function is to do anything you want to do when the device is attached to the chain, prior to getting the data irbs. And to set the device context for later data irbs.
2. `VMI_IRB_DETACH`: You got it!!! Any clean up that you need to do before the device is detached from the chain. One typical clean up is to free the memory space used to store the device context allocated during device attach. Any other clean up that is required during device detach is done here.
 3. `VMI_IRB_CONNECT`: When a connect request is sent from the connection initiating process a connect irb is passed down the send chain. The device that receive the connect irb and adds any data into the connect irb that is required for the corresponding device on the receive chain. This data is added in the form of buffer ops onto the slab.
 4. `VMI_IRB_DISCONNECT`: When a disconnect request is sent from the connection termination process, a disconnect irb is passed down the send chain. The disconnect request is received as a disconnect irb is received as a `VMI_IRB_DISCONNECT_REQ` on the other side.
 5. `VMI_IRB_CONNECT_REQ` and `VMI_IRB_DISCONNECT_REQ` are the irbs received on the receive chain in correspondence to a `VMI_IRB_CONNECT` and a `VMI_IRB_DISCONNECT` on the send chain.
 6. `VMI_IRB_SEND`: For a data send irb just put the data and send it!!! Of course it is a send irb what else you want to do with it.
 7. `VMI_IRB_RECEIVE`: Take out the data and pass it on!!! You got ur data man!!!
Enjoy.

You are essential done coding your device have fun. Hope this document helped you immensely.