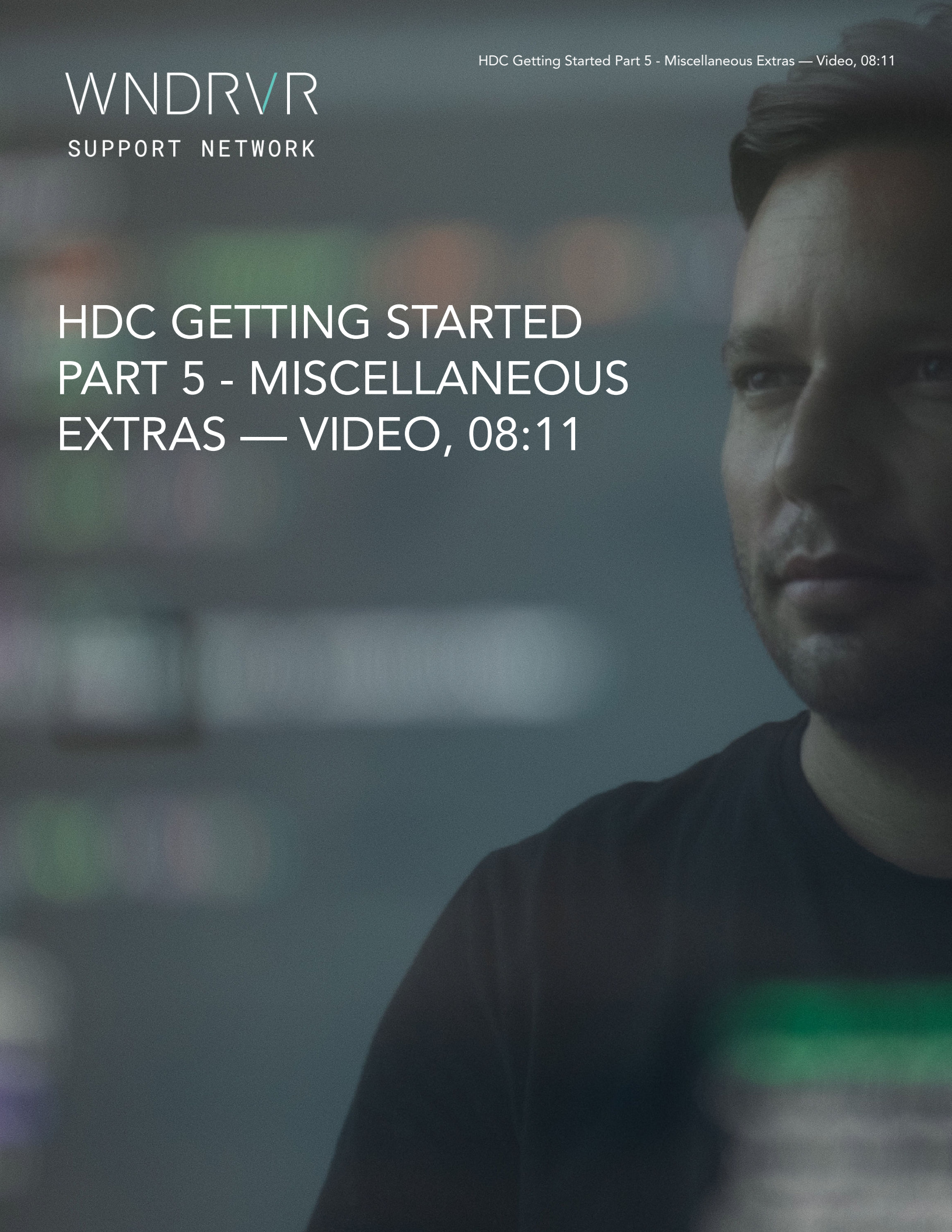
WNDRVR
SUPPORT NETWORK

HDC GETTING STARTED PART 5 - MISCELLANEOUS EXTRAS — VIDEO, 08:11



Corporate Headquarters

Wind River
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.
Toll free (U.S.A.): +1-800-545-WIND
Telephone: +1-510-748-4100
Facsimile: +1-510-749-2010

For additional contact information, see the Wind River website:

www.windriver.com

For information on how to contact Customer Support, see:

www.windriver.com/support

HDC Getting Started Part 5 - Miscellaneous Extras — Video, 08:11

29 January 2019

1. HDC GETTING STARTED PART 5 - MISCELLANEOUS EXTRAS — VIDEO, 08:11

Published on 6 October 2015

Transcript

Time (mm:ss)	Narration
From: 00:12	Ok, in this session I want to try to review some of the samples that were also provided that we haven't looked at yet. So let's go into the directory, let me see where we're at right now, we're in our home directory. So let's change directories to our /blds/code, and let's take a look down at some of these lua scripts. Of course we've already taken a look at posting data, and we have also taken a look at registering the applications that we ran in the previous examples. This is actually an older version of the script that we ran. This one only did the printfs and not posting the data item back. But I wanted to take a look also at some of these other applications like, this is how you would go about registering an alarm, and it looks exactly like registering data.
From: 01:02	You simply create the object and specify it's an alarm type instead of a data type. And then, but then, you set the value just as you would for a data type, and then post it exactly the same way. Also for events, you'll notice that again, it's essentially exactly the same thing. The only thing that's really changing is the type that you're posting here. And remember the difference between an alarm and an event, really, is that an alarm has a persisted state and requires some kind of an action. Whereas an event is really something that is just going to be recorded, and noted in the log. Also I have a script here provided for you to try, just as we did with post data from the command line, you can do post alarms directly from the command line as well. And then I have a couple other applications here, you can kind of see, this one, this is an example of going through and monitoring the available disk space,
From: 02:03	and it would actually post an alarm if the disk space got to a low condition. And so you can play with that, and try that out on your own time. Also, you can see I provided an application here, that you could, you could potentially use for doing an RPM install, so our package manage install process. And so these are just kind of some examples of some other things that you might want to do, and so you can kind of look at these later and see what they do. The other piece of code that I've shown you use of a couple of times but didn't actually explain what it was doing was the doit and the sudoit pieces. And let's go into the lua_setup directory, and lua-tools. Actually, we did look at this already. Let's go to the build directory,
From: 03:03	and down in the utils directory is what I wanted to show. And you'll notice here is the source code for the doit and the sudoit utilities that we used in the previous sessions. And so let's take a look at that real quick. These are applications that you would compile, just like you would for, any of the other applications that we've been looking at. These were implemented as C code, and we probably could have done these in lua as well. But you can see that here, in the main function, it's really just doing, it's subscribing, and it's subscribing an action, and the action is defined by, at the top of the program, called APP_NAME. And so you can see the APP_NAME is simply sudoit. And so it's registering an application with the agent, so that any time you want to invoke a function, it's called by that application name. But then you can see that all it's going to do, is once
From: 04:02	sudoit is invoked, it's going to come down here, check some arguments, and then it's simply going to do a system on the arg string. So whatever was passed in, it's really just going to invoke that as an external call out to the system. And that's it. And then it returns. And doit works exactly the same way. You can see that it's invoking system parameters as well. The only difference is one is invoking them with root permission, and one is not. And then the Makefile

	simply, if you want to rebuild those applications, you can. The binaries are provided in the build so you can just use those as-is, and they are also already pre-installed in the target image that boots up. And then one other thing I wanted to show you, in this series, was that we used a couple of actions already in the system. But those
From: 05:04	actions, I didn't describe very well how those were defined, and I want to go back here and take a look for a moment, at. When we looked at these actions, you can see that we had these mylua functions, and the register_apps functions, and those were pre-defined for us and I didn't show you how those were created. So let's go over into the Manage tab. And now let's go down and view the current actions that have been defined. And you can see that, they're defined here as "myluafunc1", "myluafunc2", and there's some other, you know functions, the "register_apps" function. But let's take a look at the mylua function because we invoke this in the target, and we had a lua script registered to perform this function, but we had to define some setup in the Helix Device Cloud platform first. And, you'll notice that, in the script,
From: 06:04	it was just printing the arguments that were passed in. And here you can see, this is where I'm defining what argument I wanted to pass in. And so the name of the function and the arguments you want to pass in, and that's really all that's necessary to invoke that function. Now you can set up some permissions on who can edit the function definitions, or you can also edit permissions on who has execute permissions on these functions. And right now I've defined them as everybody can run it. So, but anyway that's essentially how you would define the functions. But coming over here into "New" is how we could possibly create new ones. And you can see that defining a new one, you have to specify, you know, the name of the function, and then down in the target, you have to register an application that will handle that function. But then you can specify to execute an application, and that's really the secret to
From: 07:01	turning on action executions in the target. And, by giving it a name, and specifying that you execute this application; that's really all that's necessary in order to make those applications run on the target. So I created these here, from the cloud platform, and yet you can still register other applications down in the target. Now, you'll notice when we did the package definitions, and we used doit and sudoit, those were assuming that they were already defined, so we didn't create an action for those, because in the package definition, it was assuming that those were already defined for us. Ok, in the next section then, we're going to move on and talk about the application development from the host side. So, we've been looking at installing applications, and triggering and running applications, in the embedded device side connected into the agent. But now we're
From: 08:00	going to take a look at running applications from a Web application that can control this. And that will be our next section.

Contact: nlyons

Content ID: 045832

template('WindRiver/function/JS/wrConditionalContent');