isystem.connect SDK: Creating winIDEA workspace

Application Note

September 2023

Contents

Introduction

Introduction

isystem.connect SDK enables you to use BlueBox tools over different programming languages. This Application Note describes how to create, edit and save winIDEA workspace using isystem.connect with Python.

Follow the isystem.connect SDK page to be up to date with recent updates on isystem.connect SDK.
Please see isystem.connect SDK User's Guide if you are not familiar with isystem.connect SDK.

For creating winIDEA workspace with isystem.connect SDK, the following Python script examples are available:

- Basic winIDEA workspace
- Advanced winIDEA workspace

Full example

Example how to setup workspace using *WorkspaceConfigurator* is shown below.

```
import isystem.connect as ic

# When using this example, make sure to download also ws_cfg.py from the
# same location as this file.
from ws_cfg import WorkspaceConfigurator



def main():
    cmgr = ic.ConnectionMgr()
    cmgr.connectMRU('')

    wsCfg = WorkspaceConfigurator(cmgr)
    WORKSPACE_NAME = 'example.xjrf'
    wsCfg.create_workspace(WORKSPACE_NAME)
    wsCfg.set_emulator_type('iC5000')
    wsCfg.set_USB_comm('iC5000 (SN 12345)')
    #wsCfg.set_TCP_comm(IP='12.34.56.78', port='1234')
    wsCfg.set_SoC('LS1012A')
    wsCfg.add_application('myApplication0')
    wsCfg.add_symbol_file('myApplication0', 'program.elf', 'ELF')
    wsCfg.add_memory_space('memorySpace0', 'Core0', 'myApplication0')
    wsCfg.add_program_file('program.elf', 'ELF')

    wsCfg.set_demo_mode(True)

    # Download
    debugCtrl = ic.CDebugFacade(cmgr)
    debugCtrl.download()

    wsCfg.save_workspace()
    wsCfg.close_workspace()

if __name__ == "__main__":
    main()
```

Explanation

Setting up WorkspaceConfigurator

WorkspaceConfigurator is a controller that can be used to create and configure a winIDEA workspace from scratch.

How to use *WorkspaceConfigurator*:

```
import isystem.connect as ic
from wsCfg import WorkspaceConfigurator

cmgr = ic.ConnectionMgr()
cmgr.connectMRU('')
wsCfg = WorkspaceConfigurator(cmgr)
```

Creating a new workspace

A new workspace can be created by calling the `create_workspace()` function:

```
wsCfg.create_workspace(workspaceFileName='example.xjrf', workspaceDir='C:\\MyExample')
```

`workspaceFileName` selects the workspace name and `workspaceDir` selects where the created workspace will be located.

Selecting the debugger type

Debugger type is selected by calling the `set_emulator_type()` function:

```
wsCfg.set_emulator_type(Type='iC5000')
```

`Type` selects the debugger used. Available options are iC5000, iC5500, iC5700, IFX_DAS, ST_Link and XCP.

Configuring the debugger for USB

To configure the debugger for USB, the `set_USB_comm()` function is used:

```
wsCfg.set_USB_comm(device='iC5000 (SN 12345)')
```

`device` is the USB device name composed from debugger type and the debugger serial number. Used when there is more than one debugger connected to your PC, otherwise it can remain empty. This option is the same as the *Device* field in *Hardware / Debugger Hardware / Hardware Configuration / Communication*.

Configuring the debugger for TCP

To configure the debugger for TCP, the `set_TCP_comm()` function is used:

```
wsCfg.set_TCP_comm(IP='12.34.56.78', port='1234', useGlobalDiscoveryPort=False)
```

`IP` selects the IP address used, `port` selects the TCP port used, and `useGlobalDiscoveryPort` enables or disables the use of the global discovery port, which is used to discover debuggers on the local network.

Selecting the target device

Target device can be selected with the `set_SoC()` function:

```
wsCfg.set_SoC(name='LS1012A')
```

`name` selects which target device is used. The list of target devices in winIDEA can be found in *Debug / Configure Session / SoCs / Device* drop down menu.

Creating application

Application is created using the `addApplication()` function:

```
wsCfg.add_application(name='myApplication0')
```

`name` selects the name of the created application.

Adding symbol files to application

Symbol files are added with the `add_symbol_file()` function:

```
wsCfg.add_symbol_file(appName='myApplication0', path='program.elf', Type='ELF')
```

appName selects to which application the symbol file is added, *path* is the full path to the symbol file and *Type* selects the file type used. Available file types are BIN, ELF, HEX, OMF51, 25(OFM2 251/MX51), SIT(SLO text), TMSCOFF, UBROF and S37.

Creating memory space

Memory space is created with the `add_memory_space()` function:

```
wsCfg.add_memory_space(name='memorySpace0', coreName='Core0', appName='myApplication0',
isEnabled=True)
```

`name` selects the name of the created memory space, `coreName` selects which device core is associated with this memory space, `appName` references the application associated with this memory space, and `isEnabled` selects if the created memory space is enabled or disabled.

Adding program files

Program files are added with the `add_program_file()` function:

```
wsCfg.add_program_file(path='program.hex', Type='HEX', offset=0x1000)
```

`path` is the full path to the download file. `Type` selects the format of download file used. Available formats are BIN, ELF, HEX and S37. `offset` sets the download offset for the download file.

Saving workspace

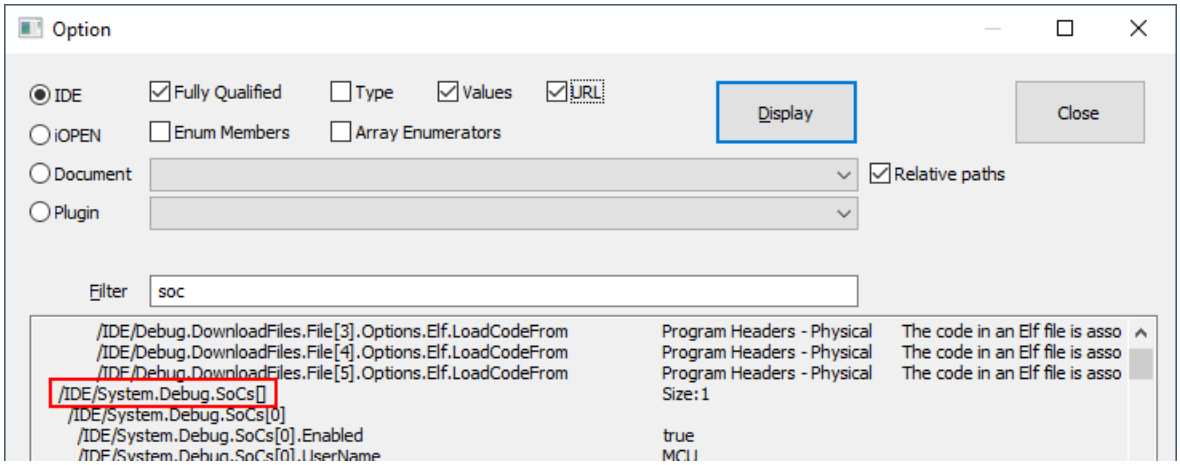
Saving workspace after implementing changes is done with the `save_workspace()` function:

```
wsCfg.save_workspace()
```

How it all works

COptionController

`COptionController` is an SDK controller that can be used to access and change winIDEA options that are not available using the standard SDK functions. To use `COptionController` you must first get the URL of the option you want to set. This can be found in winIDEA in *Help / Display Option*.



From *Display Option* dialog box, you copy the SoC list URL and create a `COptionController` instance for that list:

```
import isystem.connect as ic

cmgr = ic.ConnectionMgr()
cmgr.connectMRU('')
optSoCs = ic.COptionController(cmgr, '/IDE/System.Debug.SoCs')
optSoC = optSoCs.at(0)
```

Below is a demonstration of a few `COptionController` functions you can use to manipulate winIDEA options.

Changing value of the `/IDE/System.Debug.SoCs[0].Enabled` option to disabled using the `COptionController set()` function:

```
optSoC.set('Enabled', 'false')
```

Changing multiple options using the `set_multi()` function:

```
inParams = {}
inParams['Enabled', 'false']
inParams['UserName', 'Device0']
optSoCs.set_multi(inParams)
```

Adding another SoCs instance:

```
optSoCs.add()
```

Removing the first instance from list of SoCs:

```
optSoCs.remove(0)
```

Output number of SoCs:

```
print(optSoCs.size())
```

CConfigurationController

`CConfigurationController` is used to get COptionController objects for different commonly used URLs. Using `CConfigurationController` to get COptionController instance eliminates the need to manually search URLs and ensures compatibility between different winIDEA versions, since option URLs may change in future versions.

Creating COptionController SoCs using `CConfigurationController`:

```
cmgr = ic.ConnectionMgr()
cmgr.connectMRU('')
configCtrl = ic.CConfigurationController(cmgr)
optSoCs = configCtrl.ide_SoCs()
```

How the WorkspaceConfigurator implements its functions

This section explains the implementation of the `WorkspaceConfigurator`.

Creating a new workspace

To create a workspace, first a `CWorkspaceController` instance should be created. CWorkspaceController SDK reference:

```
import isystem.connect as ic
cmgr = ic.ConnectionMgr()
cmgr.connectMRU('')
workspaceCtrl = ic.CWorkspaceController(cmgr)
```

With the workspace Controller you can create a new workspace using the `newWS()` function, `workspaceFileName` contains the full path to the generated workspace file:

```
workspaceFileName = "C:\example.xjrf"
workspaceCtrl.newWS(workspaceFilePath )
```

Configuring the debugger

Selecting debugger type

`WorkspaceConfigurator` function: `set_emulator_type`

To configure the debugger, you first need to select the type of debugger used. The available debugger types are iC5000, iC5500, iC5700, IFX_DAS, ST_Link and XCP. Selecting the debugger type:

```
optHardware = ic.COptionController(cmgr, '/iOPEN/Hardware')
optHardware.set('Emulator',"iC5700")
```

`Emulator` parameter selects the type of debugger used.

Configuring debugger for USB communication

`WorkspaceConfigurator` function: `set_USB_comm`

To configure debugger for USB communication:

```
optCommunication = ic.COptionController(cmgr, '/iOPEN/Communication')
inParams = {}
inParams['Mode'] = 'USB'
inParams['USBDeviceName'] = "#123456"

# Set the communication options.
optCommunication.set_multi(inParams)
```

The `USBDeviceName` is the USB device name composed from the debugger type and the debugger serial number. Used when there is more than one debugger connected to your PC, otherwise it can remain empty. This option is the same as the *Device* field in *Hardware / Debugger Hardware / Hardware Configuration / Communication*.

Configuring debugger for TCP communication

`WorkspaceConfigurator` function: `set_TCP_comm`

To configure debugger for TCP communication:

```
optCommunication = ic.COptionController(cmgr, '/iOPEN/Communication')

inParams = {}
inParams['Mode'] = 'TCP'
inParams['TCPPortNumber'] = "5313"
inParams['IPAddress'] = "192.168.0.22"
inParams['IPUseGlobalDiscoveryPort'] = 'true'

optCommunication.set_multi(inParams)
```

`TCPPortNumber` selects which TCP port is used, `IPAddress` selects which IP address is used, and `IPUseGlobalDiscoveryPort` enables or disables the use of global discovery port, which is used to discover debuggers on the local network.

Selecting the target device

`WorkspaceConfigurator` function: `set_SoC`

To select the target device:

```
optSoC = configCtrl.ide_SoC('MCU')
optSoC.set('SFRName', 'LS1012A')
```

The MCU input parameter of `ide_SoC` function is the default name of the first SoC device. `SFRName` selects which target device is used. The list of target devices in winIDEA can be found in the *Debug / Configure Session / SoCs / Add or Edit / SoC / Device* drop down menu.

Creating application

`WorkspaceConfigurator` function: `add_application`

First you need to create a new application and set the name of the application:

```
optApps = configCtrl.ide_apps()
optApp = optApps.add()
optApp.set('Name','myApplication0')
```

When you create a new app instance using the `COptionController add()` function it returns an `COptionController` instance for the newly created app, which in this case is called `optApp`. With this controller you set the name for the newly created app by setting the `Name` option.

Adding symbol files to application

`WorkspaceConfigurator` function: `add_symbol_file`

To add a symbol file to the application:

```
optSymbolFiles = optApp.opt('SymbolFiles.File')
optSymbolFile = optSymbolFiles.add()
optSymbolFile.set('Path', 'program.elf')
optSymbolFile.set('Options.Type', 'ELF')
```

`Path` option sets the path to the symbol file and `Option.Type` sets the file type being used. Available file types are: BIN, ELF, HEX, OMF51, 25(OFM2 251/MX51), SIT(SLO text), TMSCOFF, UBROF and S37.

Creating memory space

`WorkspaceConfigurator` function: `add_memory_space`

To create a memory space:

```
optSoC = configCtrl.ide_SoC('MCU')
optMemSpaces = optSoC.opt("MemorySpaces")
optMemSpace = optMemSpace.add()

inParams = {}
inParams['Enabled'] = 'true'
inParams['UserName'] = 'memorySpace0'
inParams['Cores'] = 'Core0'
inParams['Application'] = 'myApplication0'
optMemSpace.set_multi(inParams)
```

`Enabled` selects if memory space is enabled. `UserName` is the user name used for this memory space. `Cores` selects which device core is associated with this memory space. Available cores can be found in the *Debug / Configure Session / SoCs / Add or Edit / Memory Spaces / Add or Edit / Location (core, SMP)* drop down menu. `Application` references the application associated to this memory space, in this case the application called `myApplication0`.

Adding download files

`WorkspaceConfigurator` function: `add_program_file`

To add a download file:

```
optSoC = configCtrl.ide_SoC('MCU')
optProgFile = optSoC.opt("DLFs_Program.File")
optProgFile = optProgFile.add()

inParams = {}
inParams['Path'] = "program.elf"
inParams['Options.Type'] = "ELF"
inParams['Options.CodeOffset'] = "0x6000"
optProgFile.set_multi(inParams)
```

`Path` option is the full path to the download file. `Options.Type` selects the format of download file used. Available formats are BIN, ELF, HEX and S37. `Options.CodeOffset` sets the download offset for the download file in hexadecimal format.

Saving workspace

`WorkspaceConfigurator` function: `save_workspace`

Saving workspace after implementing changes:

```
workspaceCtrl.save()
```