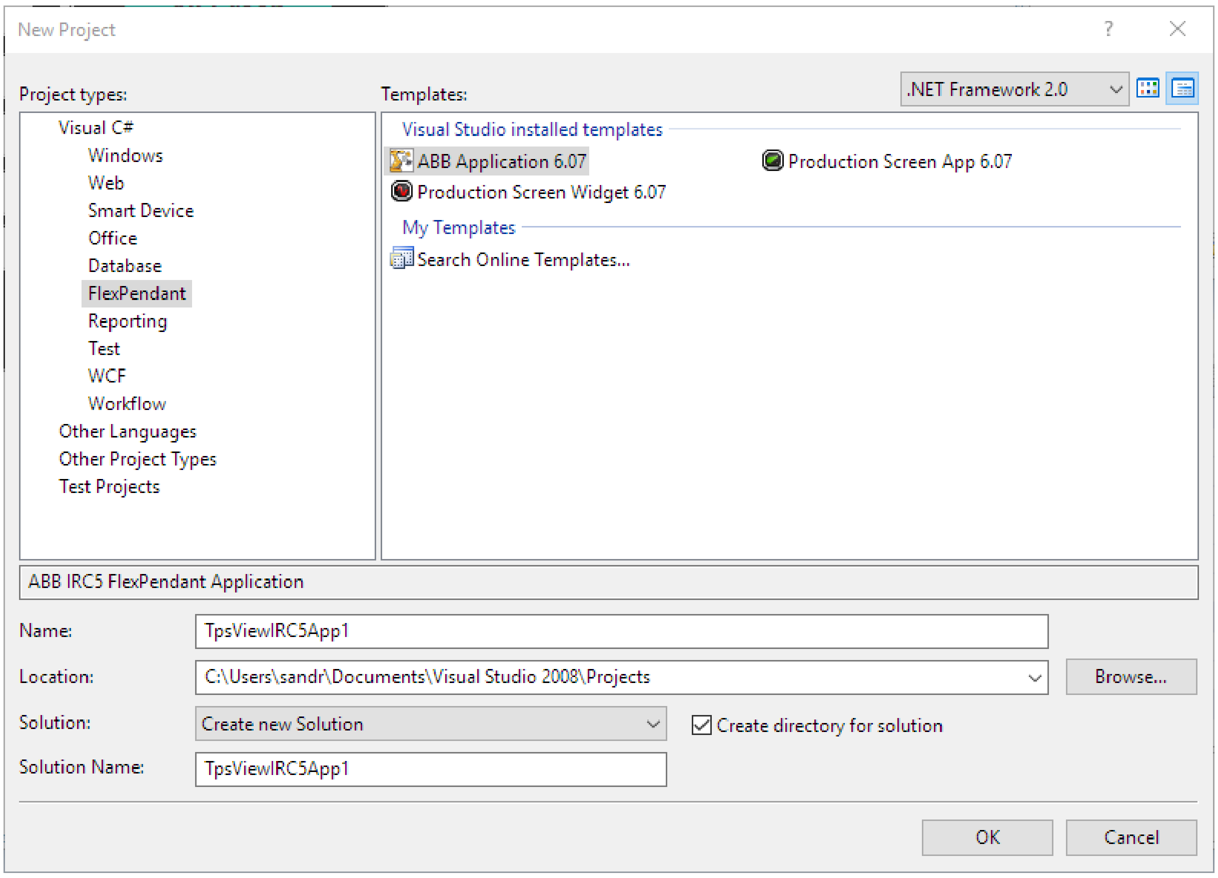
Content

Apps

Installation

In order to create flexpendant apps using Flexpendant SDK, Visual Studio is needed. VS 2008 is said to only work on Windows 7, but it works well with Windows 10 as well.

1. Install VS 2008
2. Install latest Flexpendant SDK (in case you don't already have it)
3. When adding a new project in VS 2008, templates should be available under Visual C# -> Flexpendant, see Figure 1. Make sure .NET version is 2.0, the Flexpendant does not support newer versions. If the templates don't appear, close VS 2008, open Visual Studio 2008 Command Prompt and run *devenv /installvstemplates*. Then open VS 2008 and verify that there are templates for the Flexpendant present.
4. If you want to create a new project, make sure the name starts with "TpsView". Otherwise you won't be able to build the project later.
5. When a new project has been setup, follow the instructions for setting up a new project in the Flexpendant SDK API documentation: http://developercenter.robotstudio.com/blobproxy/devcenter/FPSDK/html/bd41323c-aa49-4b28-8ab1-081d6f8b5c23.htm


New Project

Project types: Visual C# > FlexPendant

Templates: .NET Framework 2.0

Visual Studio installed templates: ABB Application 6.07, Production Screen App 6.07, Production Screen Widget 6.07

My Templates: Search Online Templates...

ABB IRC5 FlexPendant Application

Name: TpsViewIRC5App1

Location: C:\Users\sandr\Documents\Visual Studio 2008\Projects

Solution: Create new Solution

☑ Create directory for solution

Solution Name: TpsViewIRC5App1

Figure 1: Setting up a new flexpendant project

Opening project

Don't try to open a project using a newer Visual Studio, it will want to try to convert the project and then it cannot be opened with Visual Studio 2008. Instead open Visual Studio 2008 and open the project from within.

Building project

To build a project go to the tab Build and build the solution. If the build is successful it will say *Build succeeded* at the bottom left of the view. If the build is unsuccessful, the errors are listed and must be fixed before building again.

Deploying files

The files created from the build are found int the project folder in *bin/Debug*. To deploy the app to the Flexpendant, the files ProjectName.dll and ProjectName.gtpu.dll must be transferred to the home folder of the robot, possibly with file transfer in RobotStudio. To use the new app the Flexpendant must be restarted, which can be done by pressing the pen in the hole at the back of the Flexpendant or writing *fpcmd "-restart"* in Robot Commander (described in Debugging).

Lessons

In the manual for the Flexpenant SDK, http://developercenter.robotstudio.com/blobproxy/devcenter/FPSDK/html/ADA197FA-A9B1-424D-819E-3845004A4DCB.htm, there are several important parts to follow to implement a stable and reliable app.

Clean up!

It is very important to dispose of the objects, i.e. controller, rapiddata and so on. If you don't dispose of them the memory will fill up until the flexpendant crasches due to full memory. More information can be found at: http://developercenter.robotstudio.com/blobproxy/devcenter/FPSDK/html/730a47d0-0085-497a-b581-d6f5dd12acfe.htm.

Invoke

In case of a controller event, i.e. ExecutionStatusChanged. Controller events use their own thread and if a GUI thread and a controller event get into conflict, deadlocks or overwritten data may occur. A common scenario is that we want to update the GUI after a controller event has occurred. Then it is important that the GUI thread performs the changes in the interface. This can be done using Invoke. NOTE: It is not an issue when running on a virtual controller but will immediately stop working when running on a real controller. Example:

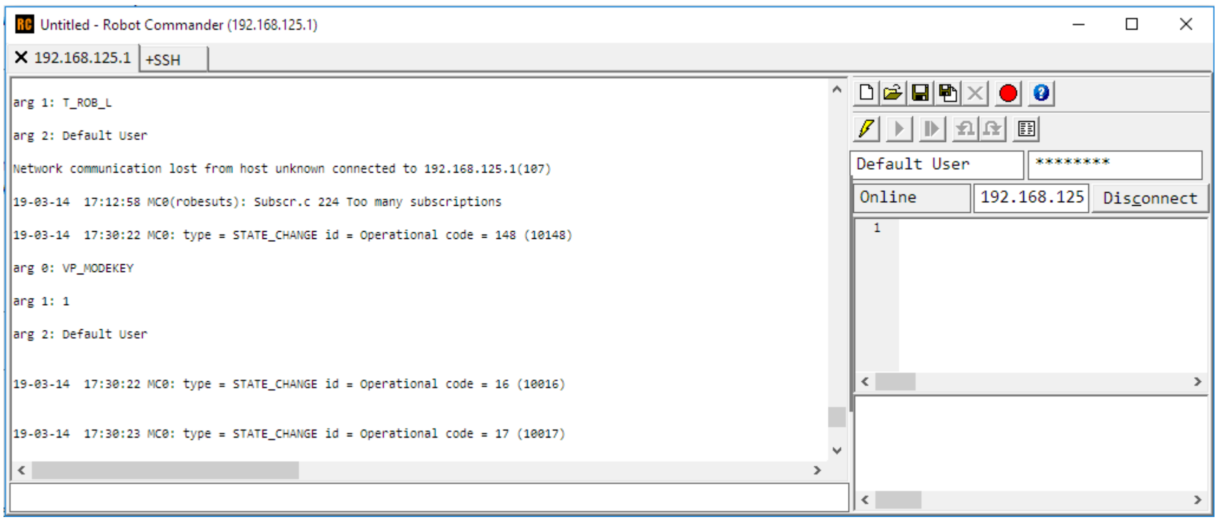
```
private void t_ExecStateChanged(object sender, ExecutionStatusChangedEventArgs e)
{
    this.Invoke(new ExecutionStatusChangedEventHandler(SetExecModeView), sender, e);
}

private void SetExecModeView(object sender, ExecutionStatusChangedEventArgs e)
{
    if (e.NewStatus == ExecutionStatus.Running)
    {
        //Perform update of GUI
    }
}
```

More information regarding reliability of a multi-threaded system can be found at: http://developercenter.robotstudio.com/blobproxy/devcenter/FPSDK/html/ae15fcc1-339b-4ef8-ba3c-6e1d0ee7462b.htm.

Debugging

One possible debugging approach is to use prints in debugging via Robot Commander. Robot Commander can be used to send commands to the robot as well.

To use Robot Commander open RobotCmd.exe and press Connect, while connected to the service port of the robot. When the connection is complete the left view will be filled with text as in Figure 2.

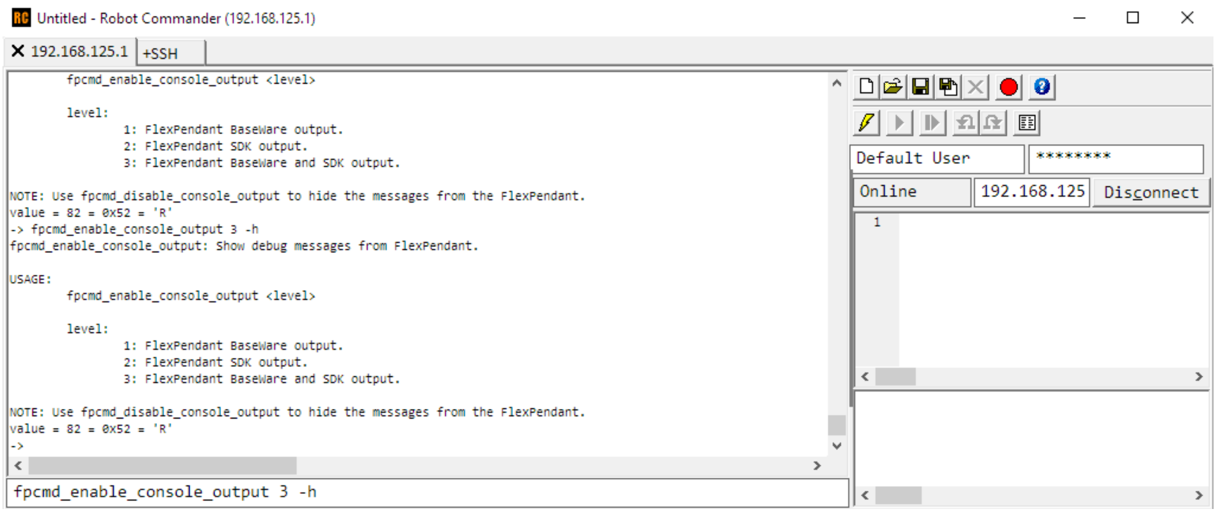


```
arg 1: T_ROB_L

arg 2: Default User

Network communication lost from host unknown connected to 192.168.125.1(107)

19-03-14  17:12:58 MC0(robesuts): Subscr.c 224 Too many subscriptions

19-03-14  17:30:22 MC0: type = STATE_CHANGE id = Operational code = 148 (10148)

arg 0: VP_MODEKEY

arg 1: 1

arg 2: Default User


19-03-14  17:30:22 MC0: type = STATE_CHANGE id = Operational code = 16 (10016)


19-03-14  17:30:23 MC0: type = STATE_CHANGE id = Operational code = 17 (10017)
```

Figure 2: Connected to the robot in Robot Commander

Some useful information and commands can be found in the Flexpendant Manual at http://developercenter.robotstudio.com/blobproxy/devcenter/FPSDK/html/82cd6e5c-eb3f-4389-b974-2ad4f4ced5f8.htm.

The important part for debugging is that the console output is enabled by writing `fpcmd_enable_console_output 3` in the command line. In case you want to know what a command does, you can add the flag -h to the end which will bring up a help section, see Figure 3.



```
        fpcmd_enable_console_output <level>

        level:
                1: FlexPendant BaseWare output.
                2: FlexPendant SDK output.
                3: FlexPendant BaseWare and SDK output.

NOTE: Use fpcmd_disable_console_output to hide the messages from the FlexPendant.
value = 82 = 0x52 = 'R'
-> fpcmd_enable_console_output 3 -h
fpcmd_enable_console_output: Show debug messages from FlexPendant.

USAGE:
        fpcmd_enable_console_output <level>

        level:
                1: FlexPendant BaseWare output.
                2: FlexPendant SDK output.
                3: FlexPendant BaseWare and SDK output.

NOTE: Use fpcmd_disable_console_output to hide the messages from the FlexPendant.
value = 82 = 0x52 = 'R'
->
```

```
fpcmd_enable_console_output 3 -h
```

Figure 3: Enabling output from the Flexpendant

To get all commands for the flexpendant write *fpcmd "-h"* in the console.

To write debug prints from the app, the Diagnostics namespace must be added to the top of the source file in Visual Studio: `using ABB.Robotics.Diagnostics;`. To print something in the program *Debug.WriteLine("");* can be used and it will appear in the console output when running the code on the Flexpendant.

Error handling

Make sure to catch any possible error event, otherwise the Flexpendant will crash. A common error event is `ResourceHeldException` which means that another application has mastership, it could be RobotStudio having write access for instance.

Other

- Transparent elements, i.e. backgrounds, are not supported on a real controller but will work fine on a virtual controller.