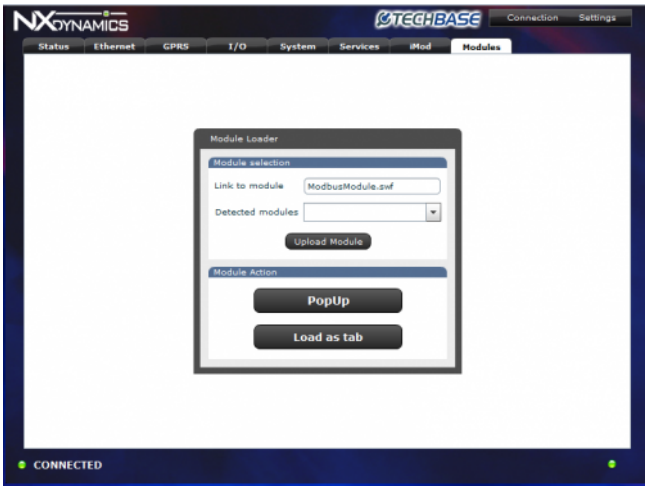
NX Dynamics - SDK

NX Dynamics Interface

NX Dynamics interface enables you to run your own modules. This feature enables users to easily design and create their graphical interfaces for executing commands on NPE. For instance, you can build a module for controlling external devices by NPE.
NX Dynamics provides API (Application programming interface), which is a simple way of communicating with NPE computer. Using the API gives you possibility to use the communication channels (Background Channel, Command Channel and Long Channel). By channels you may send commands to Telnet console for reading or interacting with NPE computer. Basic knowledge of programming is enough to create even complex modules.

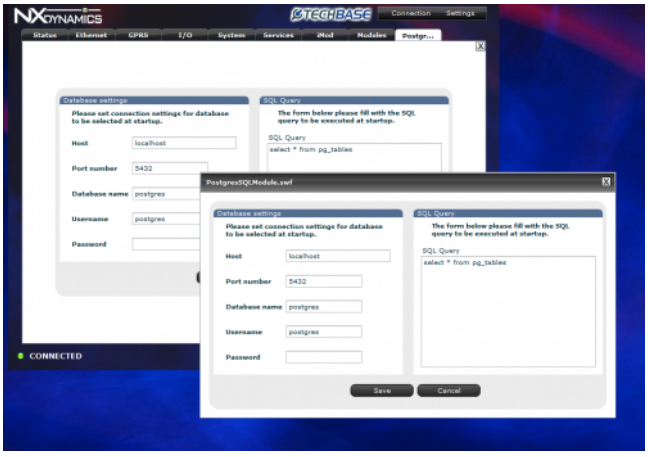
Modules Tab – loads modules that are stored on NPE.

Screen below presents *Modules* tab, which allows user to Popup modules or load them as a new NX tab.



Modules tab

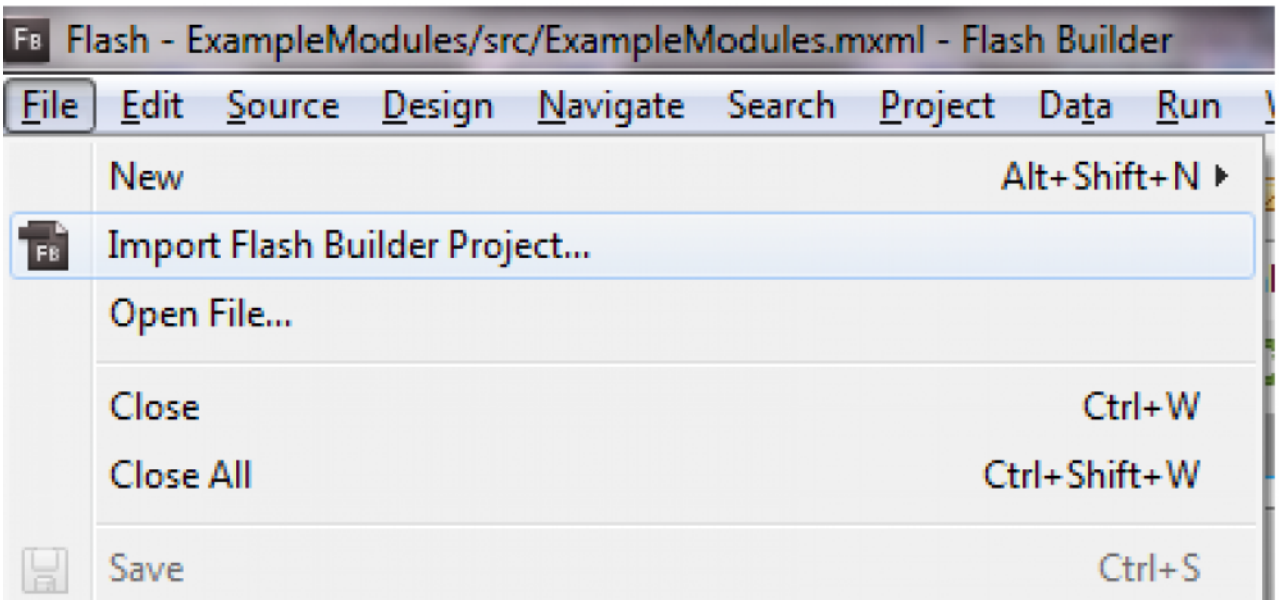
Screen below presents two ways of loading a module.



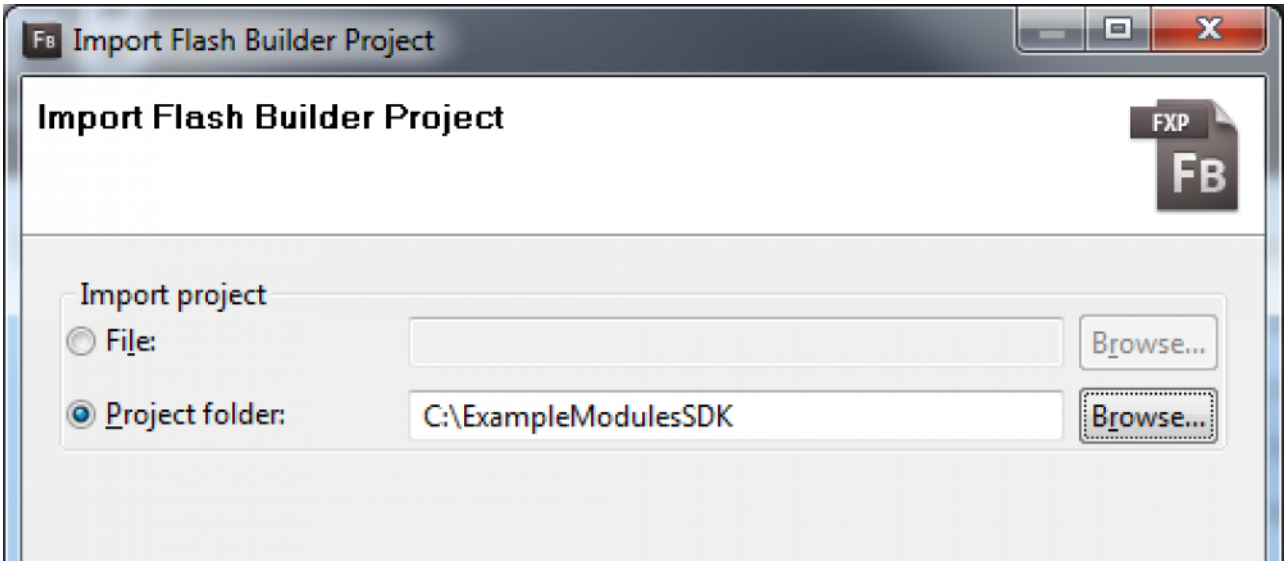
Two ways of loading a module

Configuring Flash Builder

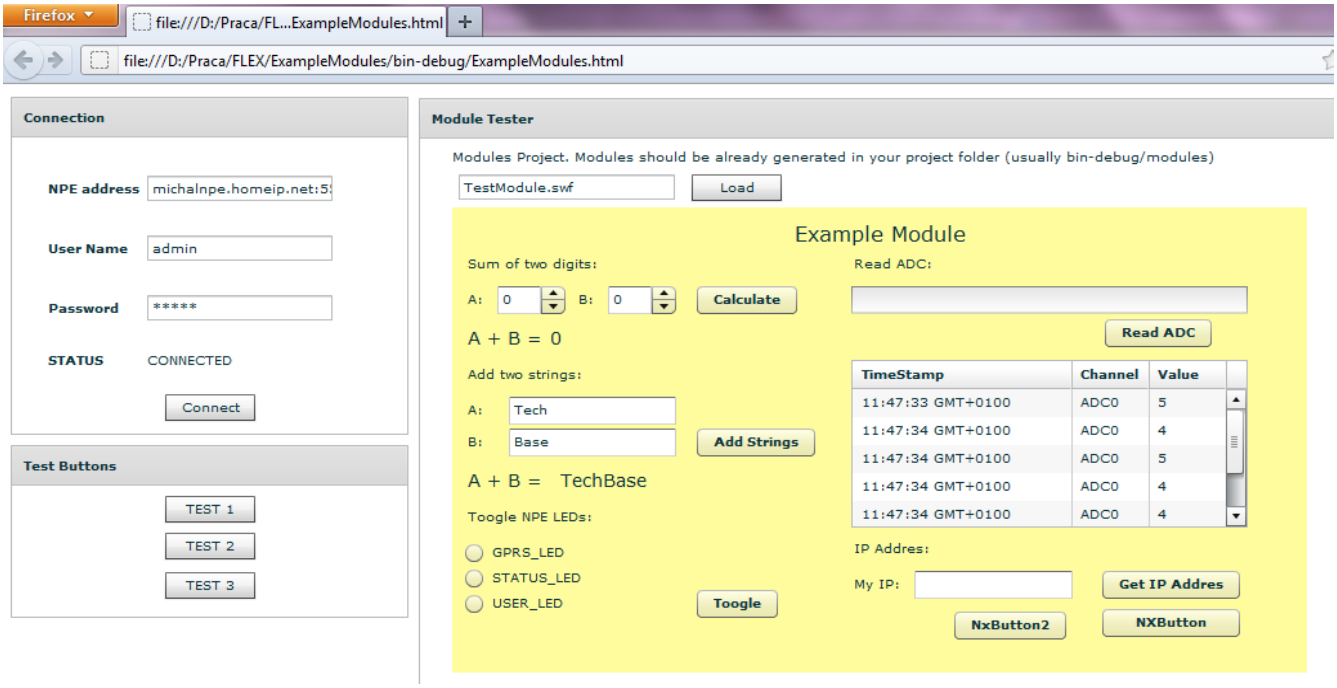
1. Install Adobe Flash Builder 4.6 downloaded from the following website: https://www.adobe.com/cfusion/tdrc/index.cfm?product=flash_builder
2. Unpack ModulesSDK.zip to any location you want. This will be your project location. Please make sure that NxStyles.css file is one directory above your imported project (ModulesSDK/../NxStyles.css)
3. After successful installation run Flash Builder, in menu go to: File→ Import Flash Builder Project...



4. Select folder with the ExampleModulesSDK you have just extracted and press OK.



5. Now you can run and debug application by pressing F11. Enter the address of your NPE, user name and password, then press connect. Now you may run your module by typing its name and pressing *Load* button.



Adobe Flash Debugger installation

Programming with debugger connected to your application is much easier. Please download the appropriate debugger to you web browser. http://www.adobe.com/support/flashplayer/downloads.html Please note that the location of downloaded files (installation files) can not be changed after installation.

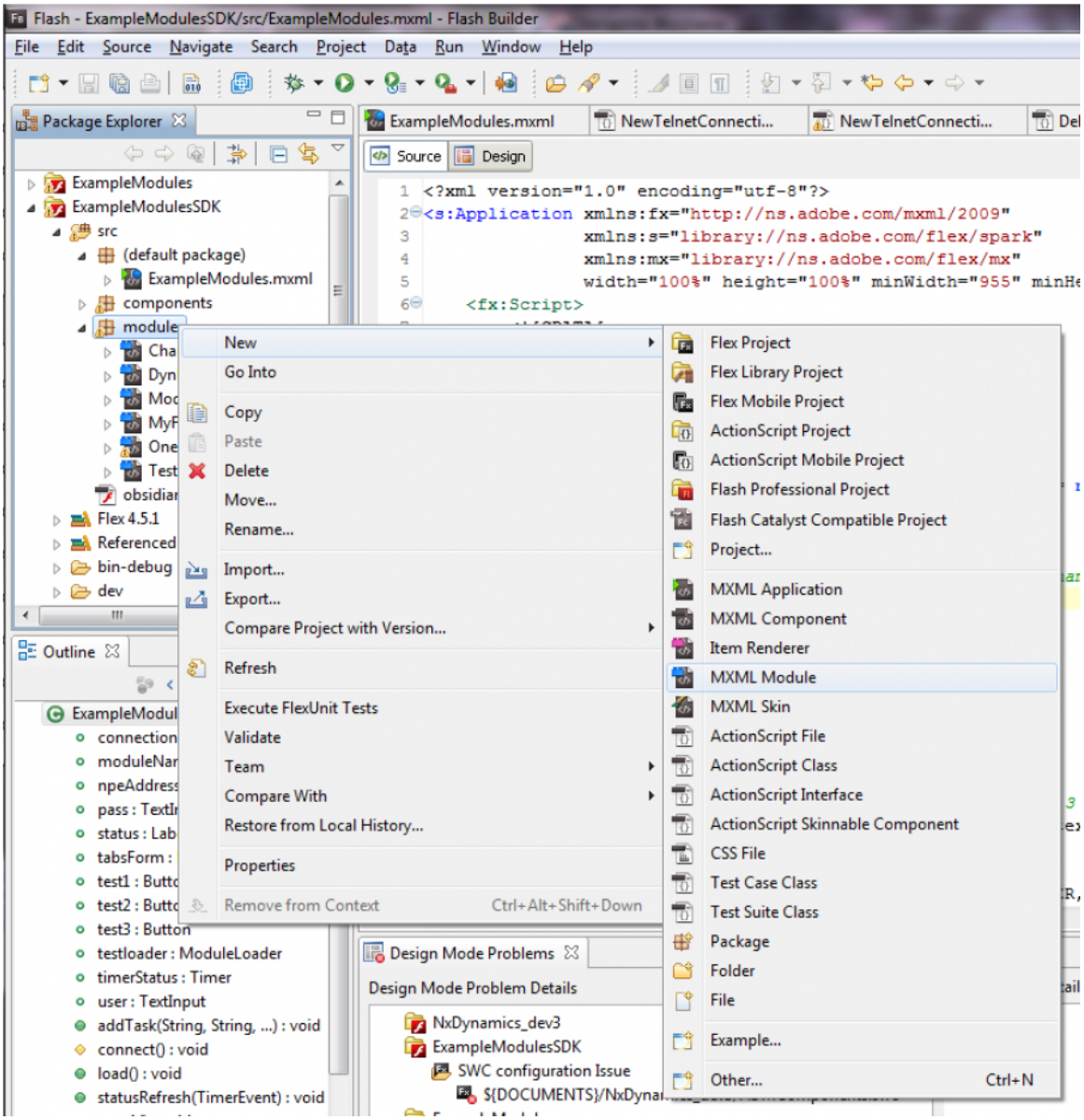
> It is highly recommended to use Firefox as a default Web browser (sometimes there are problems with connecting to the debugger via Google Chrome browser).

Default browser used for running/debugging your application might be changed in:
Flash Builder→ Window→ Preferences→ General→ Web Browser

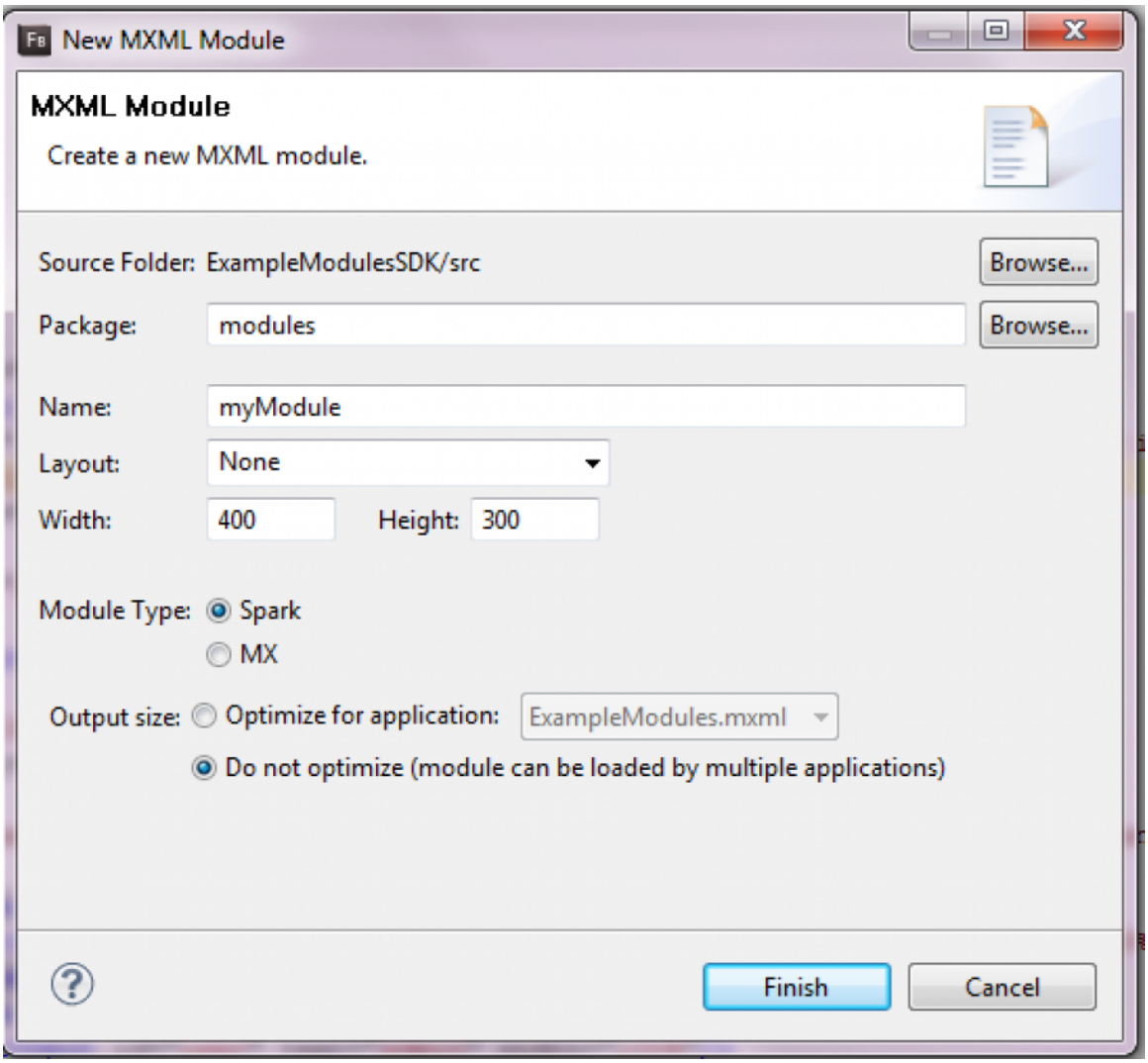
Building Modules

Building New Modules

1. After opening the project, right-click the modules package and choose the following option: *File→New→MXML Module* (see the screenshot below).

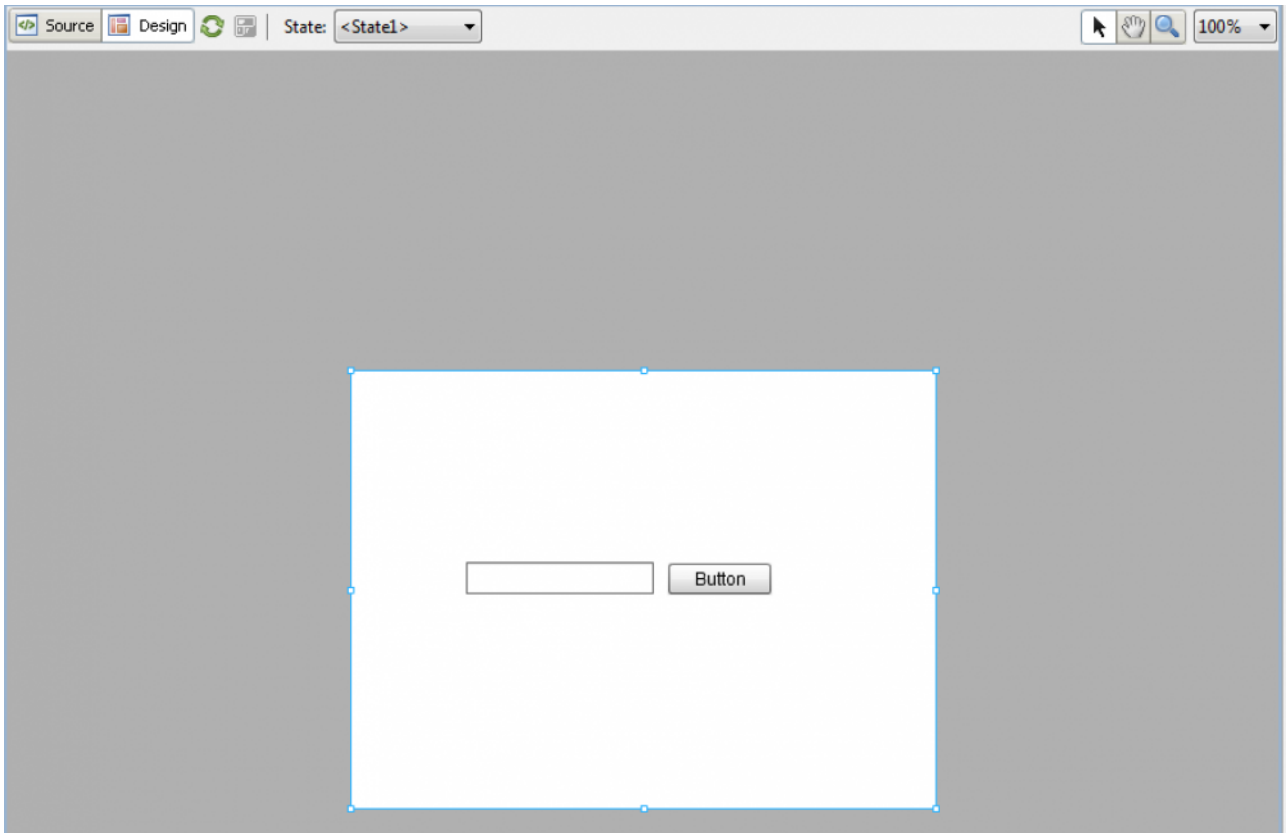


2. Enter a module name and select the „Do not optimize (module can be loaded by multiple applications)" option and click *Finish*.



3. Go to the *Design* tab which enables you to place objects graphically on your module.

4. We use TextInput and *Button* components. Drag these elements onto your module area. Then name objects (id field on the right screen) with the following names: *myData* and *myButton*.



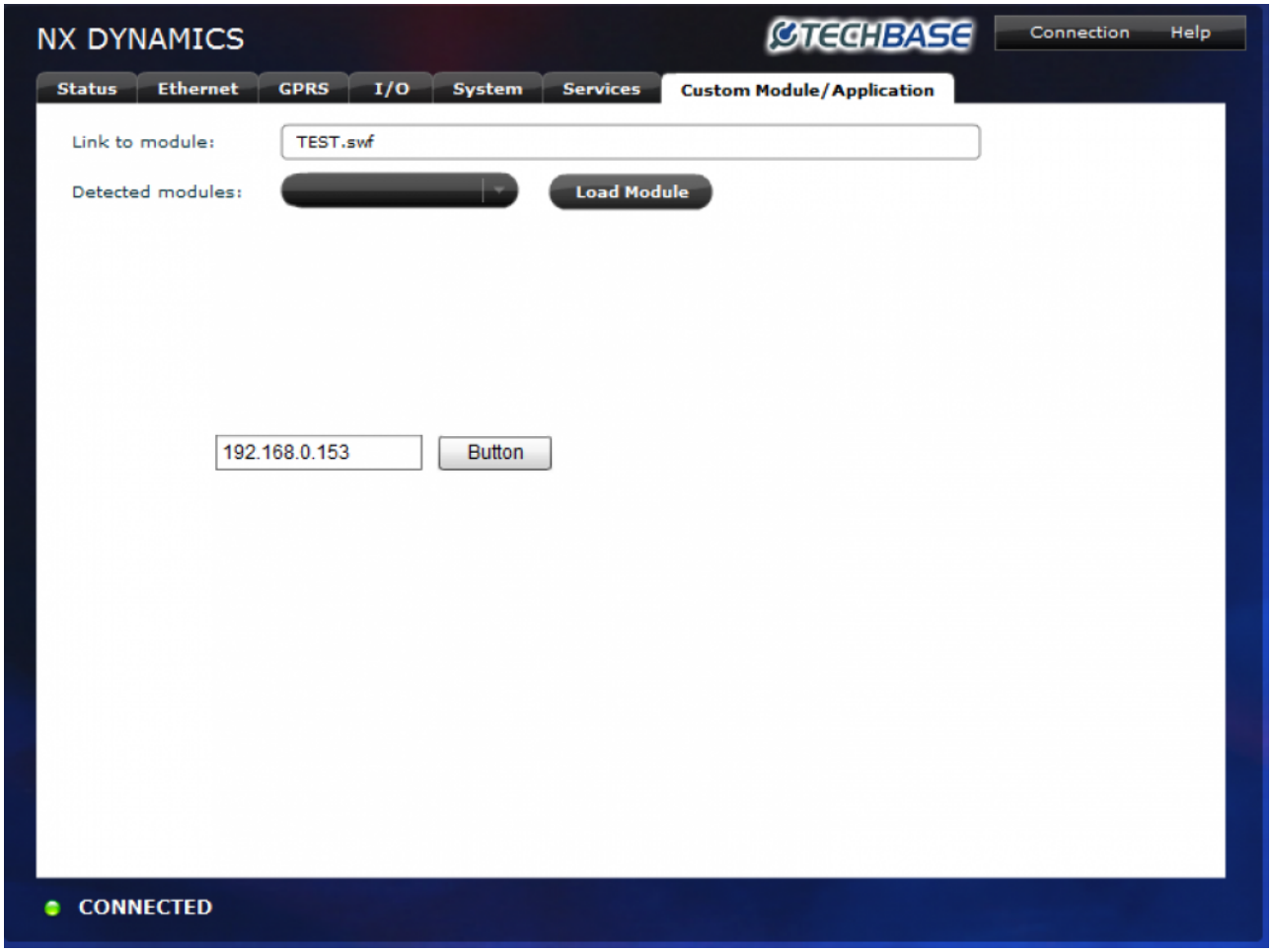
5. Now it's time to add events handler for *myButton* click event. Let's assume that we want to read NPE IP address by pressing *myButton* and display it on *myData* TextInput: right-click on *myButton* and choose the *Generate Click Handler* option. Now, define function for handling that was automatically created. (line with yellow background color were added by user):

```
...
<fx:Script>
        <![CDATA[
        import functions.*;  // remember to import this library

        protected function myButton_clickHandler(event:MouseEvent):void
        {
        var command:String = "ifconfig";
        NxGlobals.getInstance().addTask("cmd",command,{custom_function:yourActions1});

        }

        public function yourActions1(response:String):void{

        myData.text = functions.StringOperations.search(response,"inet addr:"," ",0);

        }
                ]]>
        </fx:Script>
...
```

6. Next step is to run the created module. Press F11 to start debugging, insert correct data and press *Connect*. Then type your module name - myModule.swf - and press *Load*.

Uploading Modules to NPE

1. upload the myModule.swf (which was just created in bin-debug/modules in your procject folder) file via FTP. The file has to be uploaded to /mnt/nand-user/NxModules or /mnt/sd/NxModules (the location is related to the location of NX Dynamics installation)
2. Run NX Dynamics, open *Custom Modules* tab, there should be the myModule.swf file in expandable list. Press the *Load* button.



Please remember that updating NX Dynamics will remove your uploaded modules. It is recommended to recompile modules with new NxLib included in a new NX Dynamics release.

Each module should implement **nxGeneral.NxModuleInterface** interface which forces user to define a procedure for unloading modules, showing and hiding modules (change of the NX Dynamics tabs)

```
<s:Module  xmlns:fx="http://ns.adobe.com/mxml/2009"
                   xmlns:s="library://ns.adobe.com/flex/spark"
                   xmlns:mx="library://ns.adobe.com/flex/mx" width="400" height="300"
implements="nxGeneral.NxModuleInterface">
```

Syntax Description

Function: addTask

The *addTask* function sends a defined command to *Telnet* channel and directs the response to the given function.

```
addTask(channel,command,{custom_function:yourFunction})
```

the first argument channel

- “back” stands for Background Channel
- “cmd” stands for Commands Channel
- “long” stands for Long Channel

the second argument command – Linux command to be executed on NPE

the third argument {custom_function:yourFunction} – yourFunction is the name of a function, which contains procedures for using received response.

Example:

```
var command:String = "ifconfig";
        NxGlobals.getInstance().addTask("cmd",command,{custom_function:myFunction});

        public function myFunction(response:String):void{
                Alert.show(response); // displaying response in a popup Window
            }
```

Keep in mind that:

1. callback definition should be defined as public.
2. callback function must be added without usage of parenthesis “()”

{custom_function:myFunction}); - This is some good
{custom_function:myFunction()}); - This is some wrong

Function: functions.StringOperations.search

```
functions.StringOperations.search(response,starting_sequence,ending_sequence,repetition)
Function helps to filter find the fragment of the response that is of our interest. We get
this by giving starting phrase and ending phrase.
```

```
Example:
```

response: inet addr:192.168.0.153 Bcast:192.168.0.255 Mask:255.255.255.0

```
starting sequence: “inet addr:”
```

ending sequence: “ ” (white space)

```
repetition 0 (the first)
```

functions.StringOperations.search(response,“inet addr:”,“ ”,repetition)

Result: 192.168.0.153

Modules Loaded as Read-Only

If you loaded your modules by adding modules in User Manager, you can set the *Read-Only* flag. This flag determines the permissions of the module. Please define global private variable and setter function in your module in the following way:

```
private var _RO:Boolean= false;

public function set RO(value:Boolean):void{
_RO = value;
}
```

If the module is loaded as *Read-Only* this variable will be set to true, otherwise to false. So on the basis of this parameter you can block some options in your module. Please keep in mind that _RO is not initialized just after a modulee is loaded, so during initialization event, this variable is not set yet. _RO is set while setter function RO is invoked.

Please see “Tutorial_ReadOnly.mxml” in SDK

Modules mandatory public functions

Each module has to implement 4 functions which are:

1. **inActive()** - the developer has to define procedures that are invoked when the user changed the focus to other tab in NX Dynamics. This function refers only to the modules loaded as tab in NX Dynamics.
2. **active()** - the developer has to define procedures that are invoked when the user changed the focus back on tab in NX Dynamics. This function refers only to the modules loaded as tab in NX Dynamics.
3. **unloadModule()** - the developer has to define procedures that are invoked when the user has closed the module.
4. **moduleTimeStamp()** - the developer should define the way of setting visible indicator of the module version. Please see how it's made in other modules like ModbusModule.mxml. This is recommended to assure developer that is not testing the modules loaded from web browser cache.

> If you have used Timers (flash.utils.Timer) in your module, please stop them in unloadModule() and inActive() to prevent from unnecessary operations taking place in the background.

Tutorial modules

Database access

In NX Dynamics SDK you have opportunity to easily access Sqlite3 database.

```
import communication.*;

        protected function readDatabase_clickHandler(event:MouseEvent):void
        {

        var database:DatabaseSQL = new DatabaseSQL();
        database.setDatabaseFile("/mnt/mtd/Flex/examples/example.db");
        var query:String = "select * from myTable";
        database.sendQuery(query,resultConfig);
        }
        public function resultConfig(result:Object):void{

        var str:String = result.result; // result of operation in XML format
        resultTextArea.text = str;
        }
```

Please see “Tutorial_Database.mxml” in SDK.

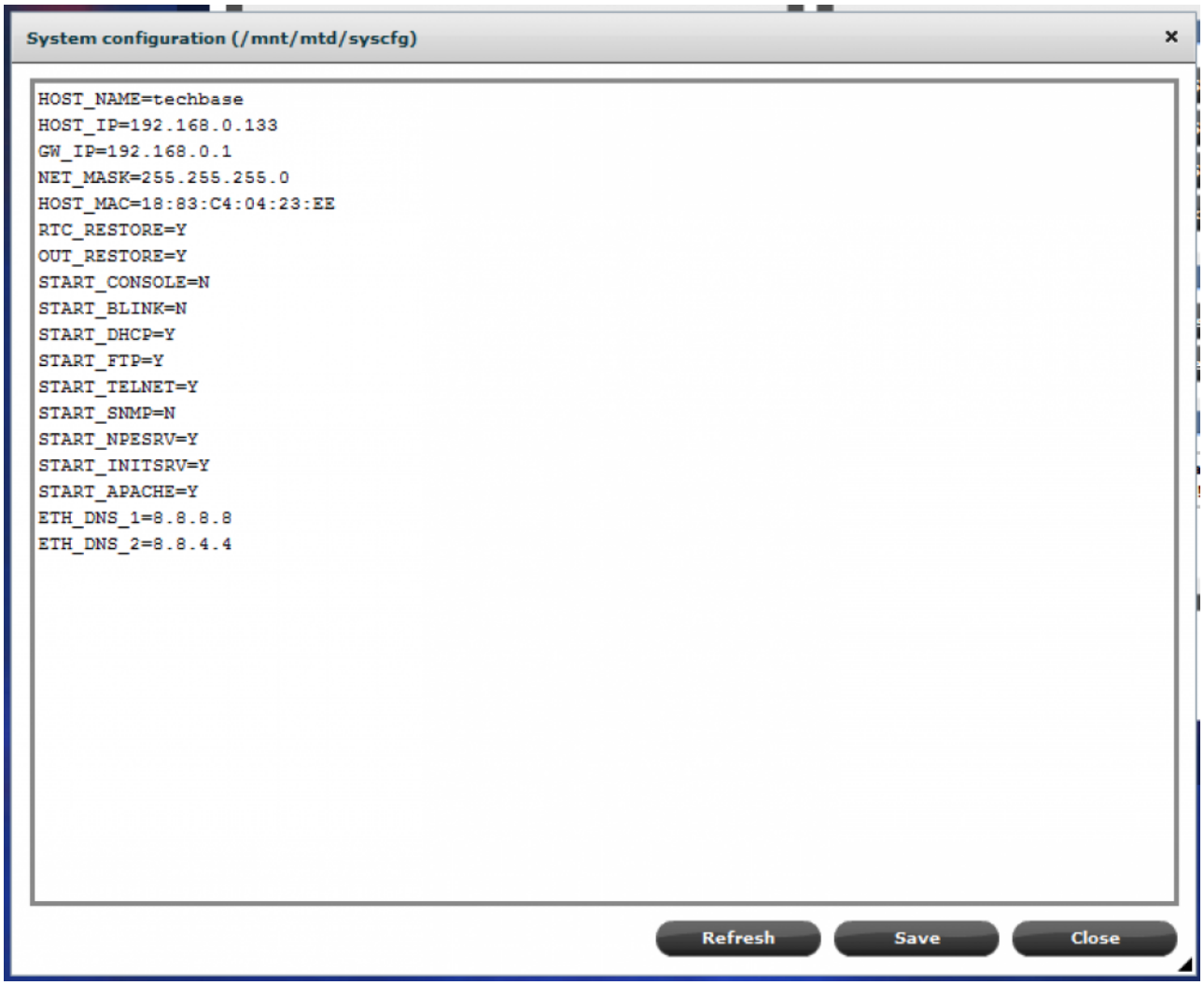
File Editor component

In NX Dynamics SDK you have opportunity to easily create files viewer/editor of files on NPE device. This may be useful for editing some configurations or controlling logs, etc.

```
var fileFileTitleWindow:FileTitleWindow = new FileTitleWindow();
PopUpManager.addPopUp(fileFileTitleWindow,this,false);
PopUpManager.centerPopUp(fileFileTitleWindow);
fileFileTitleWindow.setFile("/mnt/mtd/Flex/examples/Example.txt");
fileFileTitleWindow.mode = "tail";  // "all" for reading entire file, "tail" for readling
last lines (useful for logs)"
fileFileTitleWindow.size = "3000"; // used when "tail" mode is set. It truncates size bits
from the end
fileFileTitleWindow.setEditable(true);
```

```
fileFileTitleWindow.title = "Text Editor (/mnt/mtd/Flex/examples/Example.txt)";
fileFileTitleWindow.download();
```

Please see “Tutorial_FileEditor.mxml” in SDK



HTTP Service Object

In NX Dynamics you can easily access any remote data by using Flex HTTPService object.

Code below presents HTTPService object definition:

```
<s:HTTPService result="httpservice1_resultHandler(event)" id="request" fault=
"request_faultHandler(event)"
url="http://www.a2s.pl/npe/softmgr/Michal/wymiana/test.php" method="GET" resultFormat="text"
/>
```

Code below presents the way of sending request to a server:

```
protected function sendRequest():void
                        {
                                // SENDING REQUEST TO A WEBPAGE WITH PARAMETER
                                request.send({myParameter:"Hello World"});
                        }
```

Please see “Tutorial_HTTP.mxml” in SDK

Generating Charts Based on Downloaded CSV Files

In NX Dynamics you can easily access any remote data as it was described in the previous paragraph. In this case the basic CSV file is parsed into XY plot components.

The code below presents HTTPService object definition:

```
<s:HTTPService result="httpservice1_resultHandler(event)" id="request" fault=
"request_faultHandler(event)"
url="http://www.a2s.pl/npe/softmgr/Michal/wymiana/test.php" method="GET" resultFormat="text"
/>
```

The code below presents the way of retrieving data from response for the server:

```
protected function httpservice1_resultHandler(event:ResultEvent):void
{
   // HERE YOU CAN PROCESS INFORMATION, FOR EXAMPLE TO ADD NEW DATA TO A CHART
   // RESULT IS A STRING, SO YOU CAN USE STRING FUNCTION TO HANDLE IT

        // received response
        var result:String = event.result as String;
        outputTextArea.text = result;


        var resultArray:Array = result.split("\n"); // seperating each line (row)
        // adding random variables to chart

        for(var key:Object in resultArray){
        var columnsArray:Array = resultArray[key].split("; "); // seperating each column
(parameter)

                        // adding data to the chart
                        data.addItem(
                                {
                                yValue1:columnsArray[2],
                                yValue2:columnsArray[3],
                                yValue3:columnsArray[4]
                                });

                        }

                        }
```

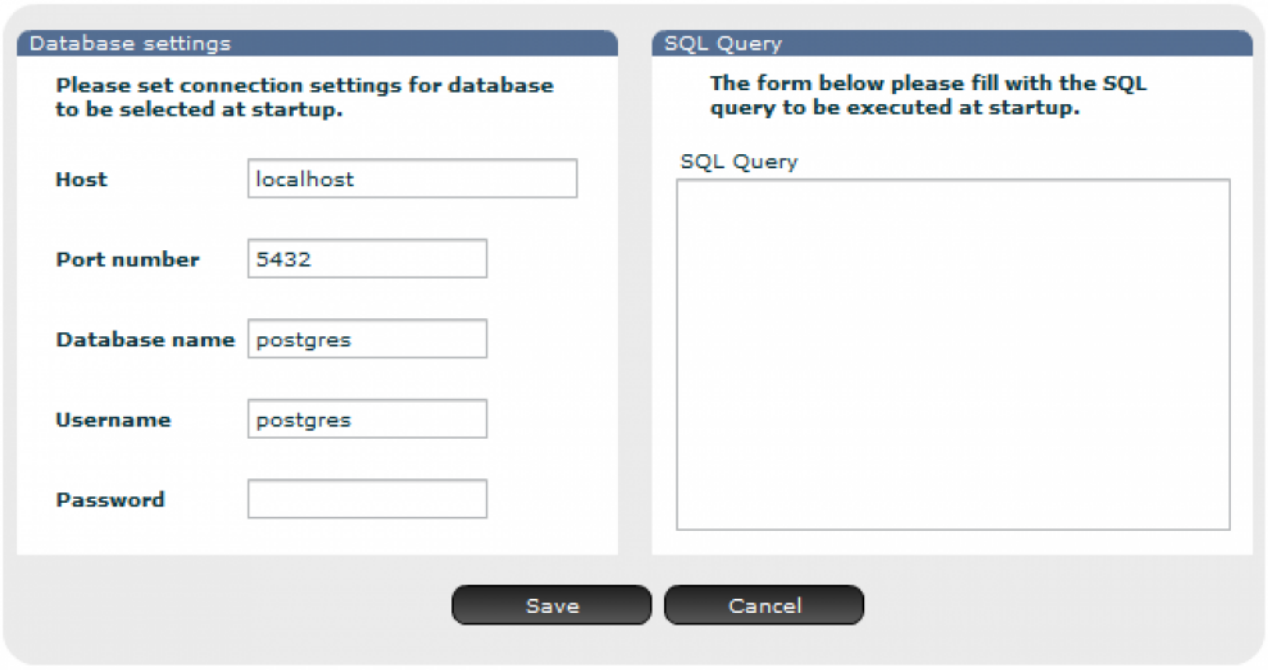
Please see “Tutorial_CSV.mxml” in SDK

NX Dynamics Modules

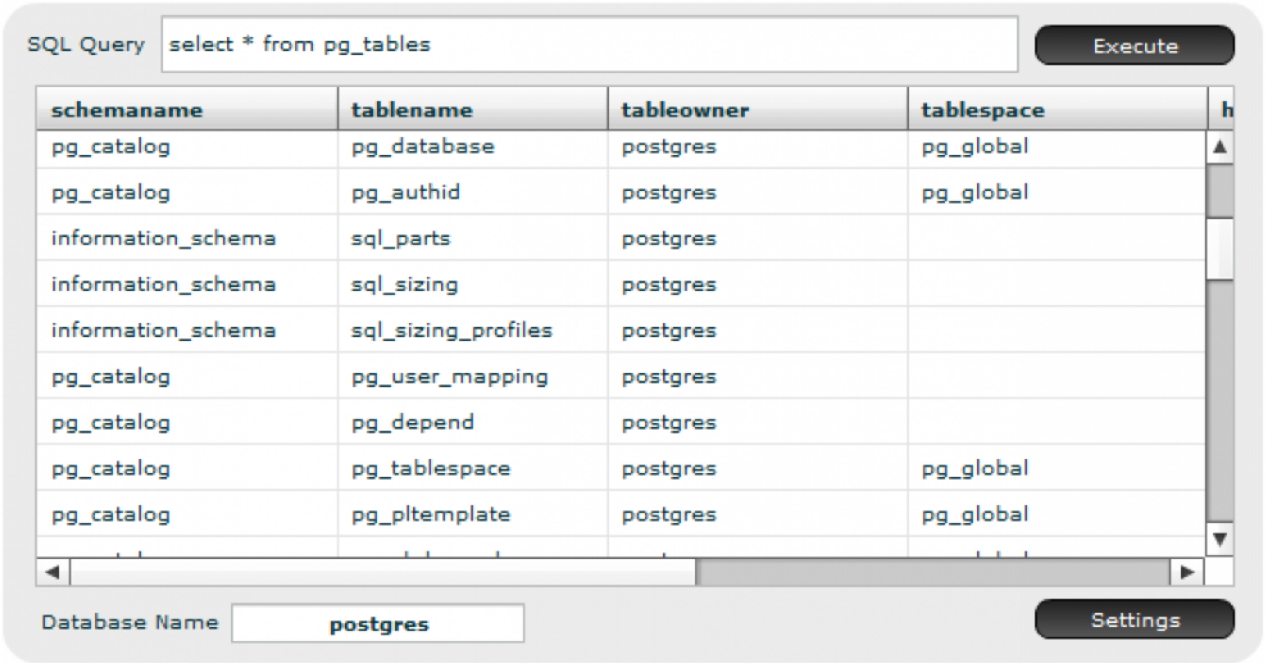
PostgreSQL Module

PostgreSQL Module can access any PostgreSQL database. You can easily define parameters of PostgreSQL connection and save them in local cache, so you no longer need to define them again. By using PostgreSQL Module you can browse your stored measurement records made by iMod. PostgreSQL records are presented in a convenient datagrid form.

Inserted parameters are stored in local SharedObject.



The way of presenting data in datagrid form



ControlPanel Modules

Before running the examplary ControlPanels please start iMod with specially defined MainConfig.xml which is provided with NX Dynamics. To run iMod with this config please just execute the following commands (it will copy iMod config and start iMod):

```
cp /mnt/mtd/Flex/ExampleXML/ExampleMainConfig.xml /mnt/mtd/iMod/
config/MainConfig.xml
imod start
```

Please note that examplary ControlPanels do not work with different MainConfig.xml

Please note that the manual for CoolingControlPanel Module refers to the SDK issued after the October 2012.

ControlPanels series is a simple and powerful way of visual representation of parameters in iMod. By editing simple templates you can easily create you own ControlPanel to monitor and control the parameters of your system.

Creating your own ControlPanel consists basicaliy of 5 steps:

1. copying the image of the scheme representing your system
2. placing viusual components onto the uploaded scheme
3. setting parameter-id for each visual component that you have placed
4. defining click action for each component
5. compiling and test the operation of your own ControlPanel

Available components in NX SDK (NxModule/sqlPackage):

- TransDoubleButton (for digital readings)
- TransDoubleButtonFill (for digital readings)
- TransDoubleButtonRect (for analog readings)

SimpleControlPanel

SimpleControlPanel Module is an easy start-up for designing your own control panels modules. Every ControlPanel consists of 2 mandatory components:

1. ControlPanel toolbox for monitoring and controlling the process of subscription to reading parameters.
2. Visual Components reading specified value of iMod parameters

Let's have a look at the one component that is placed on SimpleControlPanel.

```
<sqlPackage:TransDoubleButtonFill id="ioTest0" x="52" y="81" width="45" height="45"
                          TIGER_ID="6" toggle="true"
                          maxValue="1" minValue="0"
                          cornerRadiusVar="2" toolTip="Start/Stop main compresor in the
circuit"
                          />
```

The specific attributes that components have are:

1. TIGER_ID - its the most important attribute defining the parameter ID in the iMod that the component is going to read
2. toggle - (true/false) determines whether the clicking on the component should invoke toggle operation. If the value of parameter is 1 clicking on the component will set the value to 0

Let's have a look at another component that is placed on SimpleControlPanel.

```
                <sqlPackage:TransDoubleButtonRect id="testComponent2" x="276" y="79" width=
"99" height="42"
                                  TIGER_ID="5" click="testComponent2Click(event);"
customFunction="{doorFunction}"
                                  maxValue="1" minValue="0"
                                  label="Clickable"
                                  color="0x000000"
                  />
```

The specific attributes that components have are:

1. customFunction - this attribute sets the function that is activated while the new reading is made. You can define your own procedures here like setting custom labels depending on the current value of parameter.

CoolingControlPanel

CoolingControlPanel is a module intended to be modified by users. The purpose of this module is to give examplary live monitoring of iMod parameters. The users can program the behavior for his module.

The Other way of accessing iMod parameters

In order to read values of iMod parameters a simple SQL query might be sent to NPE. Please have a look at the **Tutorial_Database.mxml** to see the way of accessing database. The only thing that you need to do is changing the database file and query as follows:

```
database.setDatabaseFile("/mnt/ramdisk/modbus.db");
var query:String = "select TIGER_ID, Value from dane";
```

Then you can manipulate iMod parameters in XML format.

Custom Images (NX Dynamics in Web Browser)

To change images in NX Dynamics you need to change one line in NX Dynamics.html. Cabge the following line:

```
flashvars.customImages = false;
```

to:

```
flashvars.customImages = true;
```

NxDynamics.html file is located in: /mnt/mtd/htdocs/nx/

The next step is just to replace your existing images files in /mnt/mtd/htdocs/nx/images with your specific images.

Please remember that this operation requires NPE reboot, but if you want immediate effect (without rebooting) you need to perform the same operations in /mnt/ramdisk/htdocs/nx