# Java ME Porting using preprocessor directives

In-house Porting Solutions So, you've decided you do need to automate porting your application. Sure, you could outsource it, but if you're thinking about doing it in-house, you've basically got four methods from which to choose. Build one Version per Device Series: This approach is to simply develop an application for a specific series of models, for example, the Nokia Serie40 Edition 1. The problem here is that the more APIs you support, or the more your application stresses the device, the more you fragment the series since support for advanced APIs highlights the small differences within the series. For example, two similar devices will have wildly varying performance results due to the number of images on the screen.

Dynamic Detection of the Handset: This option involves testing your application during execution. For example, suppose your model is a Nokia handset. Your application would detect the device model during execution and select the appropriate behaviour depending on the model. This allows you to call methods only available on specific handsets—like fullscreen mode. You have to create one class for each specific implementation (NokiaCanvas, SiemensCanvas, and StandardCanvas). The following code demonstrates:

```
try
{
   Class.forName("com.nokia.mid.ui.FullCanvas");
   Class myClass = Class.forName("NokiaCanvas");
   myCanvas = (ICanvas)(myClass.newInstance());
}
catch (Exception exception1)
{
 try
 {
  Class.forName("com.siemens.mp.color_game.GameCanvas");
  Class myClass = Class.forName("SiemensCanvas");
  myCanvas = (ICanvas)(myClass.newInstance());
 }
 catch (Exception exception2)
 {
  myCanvas = (ICanvas) new StandardCanvas();
 }
}
```

You basically create an interface, Icanvas, and three implementations, one for Nokia devices, one for Siemens devices, and another one for standard MIDP devices. Then you use Class.forName in order to determine whether a proprietary API is available. If no exception is thrown, you use the NokiaCanvas. Otherwise, it means the current device doesn't support this API. In this case, you test another API (for example, Siemens). If another exception is thrown, it means you have to use the standard canvas.

Using a Preprocessor: Using a preprocessor, your source code will be automatically activated or deactivated depending on certain conditions. For example, to set the full screen mode on a Nokia device, you have to extend FullCanvas, not Canvas. On a MIDP 2 device, you have to call setFullScreenMode. On a MIDP 1 device, this isn't possible, so you stay in a non-fullscreen mode.

```
//#ifdef NOKIA
   extends com.nokia.mid.ui.FullCanvas
//#else
   extends Canvas
//#endif
   {
...
//#ifndef MIDP2
       setFullScreenMode(true);
//#endif
```

A preprocessor processes this source code, then you set the directives. So, to generate the application for a Nokia device:

```
//#define NOKIA
```

The preprocessor produces:

```
   extends com.nokia.mid.ui.FullCanvas
   {
```

For a MIDP 2 device, (“//#define MIDP 2”), it produces:

```
   extends Canvas
   {
      setFullScreenMode(true);
```

This solution allows for one body of source code to be adapted to each device model. You need only develop to the reference source code, including the directives. All other modifications made to the processed files will be lost after the next preprocessing. Though this solution relies on the old concept of preprocessing, this is the only technique open-ended enough to solve all the problems you'll encounter trying to port to multiple device models.


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.