Developing Mobile Apps Using Cross-Platform Frameworks: A Case Study

Shah Rukh Humayoun, Stefan Ehrhart, and Achim Ebert

Computer Graphics and HCI Group,
University of Kaiserslautern,
Gottlieb-Daimler-Str., 67663, Kaiserslautern, Germany
{humayoun,ebert}@cs.uni-kl.de, s_ehrhart@gmx.net

Abstract. In last few years, a huge variety of frameworks for the mobile cross-platform development have been released to deliver quick and overall better solutions. Most of them are based on different approaches and technologies; therefore, relying on only one for using in all cases is not recommendable. The diversity in smart-devices (i.e. smartphones and tablets) and in their hardware features; such as screen-resolution, processing power, etc.; as well as the availability of different mobile operating systems makes the process of mobile application development much complicated. In this work, we analyze few of these cross-platform development frameworks through developing three mobile apps on each of them as well as on the native Android and iOS environments. Moreover, we also performed a user evaluation study on these developed mobile apps to judge how users perceive the same mobile app developed in different frameworks and environments, from the native to the cross-platform environment. Results indicate that these frameworks are good alternative to the native platform implementations but a careful investigation is required before deciding to check whether the target framework supports the needed features in a stable way.

Keywords: Cross-platform development, mobile apps, iOS, Android, smart-device, smartphone, tablet, user evaluation.

1 Introduction

The rate of smartphones amongst cell-phones was expected to exceed the 50% boundary in the year 2012 [11] with the amount doubling each year [3]. Nowadays smart-devices, which include smartphones and tablets, are a vital platform for people to access services in their daily life, not only in developed countries but in developing countries too [2]. Due to great variations in smart-device types (from mobiles to tablets), in their hardware (different screen sizes, resolutions, and computation power), and in the underlying operating systems (e.g., Android, iOS, Windows Phone 8) make it a big challenge for software developers to develop applications (called *mobile apps* or just *apps*) for them. Developing mobile apps separately for each platform or device is costly and time consuming process while keeping focus on just one platform or device reduces the number of accessible users. This problem leads to a solution where the mobile apps are developed through frameworks, called *cross-platform development frameworks*. In these frameworks, the apps are developed just once and then can be deployed on those platforms and devices that are supported by the underlying framework. However, one of the main problems the industry is facing nowadays in this solution is that the apps developed on these frameworks normally provide not as good interaction and functionalities compared to the apps developed on the native development environments.

In last few years, plenty of frameworks for mobile cross-platform development have been released to deliver overall cost-effective and better solutions. Most of these frameworks use different underlying approaches and technologies; therefore, relying only on one for using in all cases is not recommended. In this work, we analyze few of these cross-platform development frameworks through developing three apps on each of them as well as on the two most widely used native environments, i.e., the Google Android and the Apple iOS. Moreover, we also performed a user evaluation study on these developed apps to judge how users perceive the same app developed in different frameworks and environments, from the native to the cross-platforms.

The remainder of this paper is structured as follows. In Section 2, we highlight the background. In Section 3, we describe the three scenarios for developing apps and details of the development of these apps in different frameworks and environments. In Section 4, we provide details of the user evaluation study. In Section 5, we analyze from the software evaluation perspectives. Finally, we conclude in Section 6.

2 Background

Smartphones and tablets are getting more and more popular since after launch of the Apple iPhone and iPad even though the first smartphone, the IBM Simon, was built in 1992 and then released in 1993 by BellSouth [1]. Nowadays, a number of operating systems from different vendors are available for these smart-devices. Few examples of the most famous ones are Google Android, Apple iOS, Microsoft Windows Phone, Symbian OS, and RIM Blackberry OS. Developing mobile apps separately for each platform is quite costly as it needs the same number of development time for each target platform. Moreover, it also makes the maintenance more costly and time-consuming. To resolves these issues, many cross-platform development frameworks have been developed in which developers write code once and then the resulting app can be deployed on different platforms and environments. Each of these frameworks targets a number of different platforms and environments starting at least from two. A study by Vision Mobile [8] pointed out over one hundred different mobile cross-platform frameworks. The availability of these frameworks on one-side gives developers the freedom to choose amongst them according to their requirements, but on the other side also makes it difficult to choose the right one.

Many people from the industry and academia have already analyzed and categorized the available cross-platform development frameworks. An example form the industry is a report by Vision Mobile [8] that discusses the tools for mobile cross-platform development. In academia, some studies [4, 9, 10] have also been done that analyze the frameworks and tools for mobile cross-platform development. Each of these studies analyzed different frameworks and tools and highlighted their advantages and disadvantages. They also tried to find out the better option amongst the selected ones against their chosen criteria.

In comparison to the previous work on the issue of cross-platform mobile development, the work behind this paper differs from two aspects. First, we implemented several sample applications derived from the interaction schema of mobile environment. Secondly, we verified the results through a user evaluation study to find out if users from different backgrounds are even satisfied enough with the cross-platform solutions, in order to make these solutions worth considering in the first place. The frameworks in which we were interested are: Appcelerator Titanium [5], which is an open source (under Apache License) runtime interpreter using HTML and JavaScript; RhoMobile Rhodes [6], which is a Ruby and HTML based open source (under MIT license) runtime interpreter; and MoSync [7], which is a C++ based open source (under GPL) mixture of a runtime interpreter, a source code translator and a web-to-native wrapper.

3 The Testing Scenarios and The Developed Apps

We choose three frameworks out of a variety of frameworks as well as the native iOS and Android environments for developing apps' versions and then for detailed testing based on our targeted criteria. We did not take into account the web app frameworks due to their reduced functionality as well as the app factory frameworks. Appcelerator Titanium [5] and RhoMobile Rhodes [6] were selected as these cross-platform development frameworks are based on completely different web programming languages. While the third framework MoSync [7] was selected because it combines the three approaches of hybrid frameworks and supports a number of platforms.

The sample scenarios (i.e. *MovePic, BubbleLevel,* and *AnnotatePic*) for developing the apps were derived from the perspective of interaction-schema of smart-devices, which mostly consists of touch-events (such as *tap, drag, pinch*, etc.), accelerometer, localization services, camera access, file system access, etc. Our focus was towards touch-events, accelerometer, camera access, and the file system access. We left other interaction-schema elements, such as localization services, due to their behavior complexity. For each scenario, we developed apps using the above-mentioned frameworks as well as the native iOS and Android development environments. The developed apps for the scenarios target towards not only the smartphones but also the tablets in order to consider the scalability issues. Following subsections provide the description of each scenario and the corresponding developed app's versions.

3.1 The *MovePic* Scenario

The *MovePic* scenario's goal was to test the support and processing of multi-touch gestures in smart-devices. This was done by hardcoding an image through the image-view in the underlying device. The presented image could be manipulated with multi-touch gestures such as panning, pinch zooming, rotating and dragging.

The native iOS version was developed using the built-in gesture recognizers in which the attached gestures can only be used inside the image, because in iOS the gesture recognizers are assigned to the image view only. At the time of the development, Android native environment did not have built-in gesture recognizers for multi-touch gestures, needed for this app; therefore, the functionality was realized through the manual implementation. This was done through the *OnTouchListener* interface and its *onTouch* method, which receives all the touch events recognized by the screen. MoSync did not support the rotation of elements inside a native or a web view, so an OpenGL view was used for the implementation. Due to OpenGL 3D environment and the usage of a different coordinate system than the screen coordinate system, the translation of the image was not as accurate compared to the native implementations. Appcelerator Titanium provided modules to enable multi-touch and gesture recognizers for iOS only. Therefore, the scenario was implemented only for the iOS using the free gesture recognizer module. At the time of development, the RhoMobile Rhodes lacked the handling of multi-touch gestures and the accessing of accelerometer. Hence, we skipped the first two scenarios for it. Figure 1 shows a screen-shot of the *MovePic* app, developed in the native iOS environment.

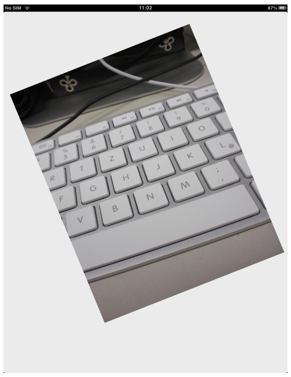


Fig. 1. The MovePic app (developed in the native iOS environment) on iPad 2

3.2 The *BubbleLevel* Scenario

The *BubbleLevel* scenario simulates a spirit level on the screen using the accelerometer data. A spirit level is a tool to measure if some object is parallel or orthogonal to the ground. The scenario's goal was to test the support and processing of the accelerometer in the device as well as checking its request rate. All the developed versions of this scenario have three different screens; i.e., a round spirit level (as shown in Figure 2.a) and a one-dimensional rectangular for vertical and horizontal (as shown in Figure 2.b and Figure 2.c respectively); which switch according to the device orientation. The round spirit level is displayed when the device lies flat. This provides a two-dimensional surface leveling. Tilting the device over 45 degree in any direction displays the vertical or horizontal bars, which are one-dimensional rectangular shapes. The accelerometers in smart-devices return three values representing the gravity vector in the device's left-handed coordinate system. The developed versions used these three values for simulating the desired spirit level.

The native iOS environment drawing functions were used to outline and to fill the circle and rectangles in all three views of the app for the native iOS implementation. The native Android version used a nested thread class for drawing the circle and the rectangles, which resulted a quicker drawing compared to the native iOS version. The MoSync's functions for drawing shapes were used to draw the circle and rectangles. While the simulating of spirit level was implemented using a Moblet, which provides interfaces for getting from the accelerometer and then using it accurately. The MoSync version looks quite similar to the native versions. The Appcelerator Titanium version used a web view showing a HTML5 document for realizing the UI of the app, as the framework itself did not have the functionality of drawing UIs. The accelerometer values were passed to the web view through events. A square was drawn instead of the circle due to the slow processing of the circle drawing.

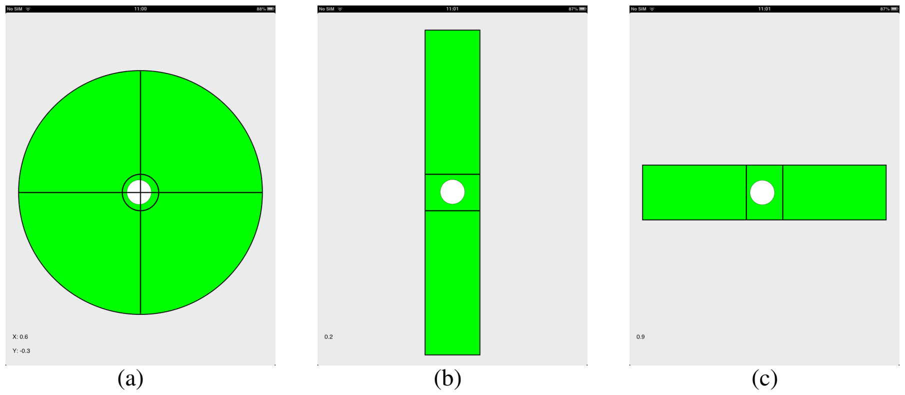
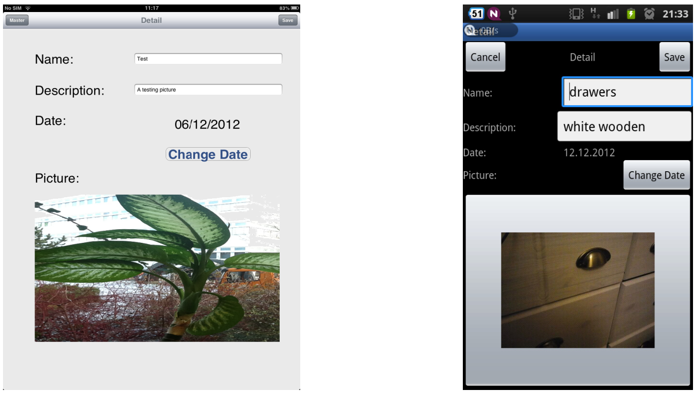


Fig. 2. BubbleLevel app on iPad 2: *(a)* a round spirit level when the device lays flat, *(b)* one-dimensional vertical spirit level, and *(c)* one-dimensional horizontal spirit level

3.3 The *AnnotatePic* Scenario

The *AnnotatePic* scenario's goal was the utilization of complex interaction-schema such as camera access, file system access and phone-specific services. The basic task was to take image from the device's camera, adding some additional information (name, description and date), and then saving the image and the data in the device's internal memory. The main screen of all the developed versions contains a list-view for showing all saved images in small-view along with their names, while the detailed screen consists of information-taking form and an image-taking button. Figure 3 shows the detailed-screen view on iPad 2 and on Samsung Galaxy S Plus.

The native iOS implementation used the *master-detail* template of iOS, which generates an app with two screens. The captured images are saved directly in the document folder of the app, while their data is saved using the core data framework of the iOS. The native Android implementation was done through a set of classes. These classes provide functionalities for generating the database, managing memory variables, saving and loading images, and managing activities for the two views. The MoSync version consisted of two sample apps for testing the camera functionality and the *master-detail* functionality with the data storage. Due to some technical issues in MoSync iOS version, only the MoSync Android version was evaluated in the user study. The Appcelerator Titanium versions were implemented through the *master-detail* application with database access. Due to the technical difficulties occurred at the IDE level, both versions were not included in the user study. The RhoMobile Rhodes targets towards data-driven business apps, so frameworks' automatically generated components were useful for the scenario. But, it also takes time to get rid of the extra-generated components. RhoMobile Rhodes custom data models in addition to own defined data models were used to implement the scenario. Due to the technical difficulties in image storing, the implementations were not included in the user study.



Name: Test
Description: A testing picture
Date: 06/12/2012
Picture:

Name: drawers
Description: white wooden
Date: 12.12.2012
Picture:

Fig. 3. Detailed-screen view of the *AnnotatePic* app on iPad 2 (*left-side)* and on Samsung Galaxy S Plus *(right-side)*

4 The User Evaluation Study

We performed a user evaluation study in a controlled environment, where the focus was on checking the interaction response of different versions of the three mentioned scenarios using different devices and operating systems. The purpose was to analyze whether normal users from different backgrounds feel any difference if the same app is presented to them, even though it is built through different frameworks and for different platforms.

The Experiment Settings

The test devices were a Samsung Galaxy S Plus and a Samsung Galaxy Tab for Android, and an iPhone 3GS and an iPad 2 for iOS. The tested versions were 12 implementations of the three scenarios for both kinds of devices (smartphones and tablets). Six of these versions were developed in native environments (i.e., Android and iOS). The remaining six implementations were: the MoSync implementations of MovePic for Android and iOS, the MoSync implementations of BubbleLevel for Android and iOS, the Appcelerator Titanium implementation of BubbleLevel for Android only, and the MoSync implementation of AnnotatePic for Android only. Due to the technical difficulties in other developed versions, we did not include them in the study. Table 1 shows those versions that were evaluated in the user study.

Table 1. The tested versions in the user evaluation study

	Native platforms		MoSync		Appcelerator Titanium	
	Android	*iOS*	*Android*	*iOS*	*Android*	*iOS*
MovePic	✓	✓	✓	✓		
BubbleLevel	✓	✓	✓	✓	✓	
AnnotatePic	✓	✓	✓			

The User Groups and The Experiment Layout

We performed the evaluation study with 9 users (2 females, 7 males). We categorized them according to their experience with smart-devices where 3 were expert in using Android based devices, 3 were expert in using iOS based devices, while the remaining 3 had no experience at all with smart-devices. The age of users was between 23 and 31 years old with a mean of 25.88. For each tested version, users were asked to judge the interaction-response time and the overall satisfaction with the app on a scale from 1 to 5. After testing different versions of the same scenario, each user was asked to name the version they would prefer for future use with multiple answers possibility.

Results and Discussions

The conducted user evaluation study provided some interesting results that could be useful for deciding the right environment(s) for developing mobile apps. In the MovePic scenario, the native iOS implementation received the best results (as it would be used by 7 users out of 9 users, see Figure 4) but the native Android and MoSync implementations too achieved comparable and good results. The reason that most users preferred iOS version was probably due to the usage of iOS built-in gesture recognizer in the implementation, which provided a better interaction response than the others. Moreover, users who were expert with smart-devices preferred the MoSync version of their own used platform instead of the other one, although both MoSync versions were identical.

In the BubbleLevel scenario, the MoSync Android version received the best results, as 7 users preferred it (see Figure 4), while the MoSync iOS and Android native versions received well appraisals too. On the other side, the iOS native and Appcelerator Titanium versions received just moderate ratings as none of the users showed any interest for using them in future. One thing that needs to be considered while comparing an app having the accelerometer functionality in Android-based devices is the quality of accelerometer in the underlying devices, as the device with high power processor might give better output than the others.

In the AnnotatePic scenario; the native iOS and Android versions received slightly better estimation in terms of the overall satisfaction, while in the interaction response time all the three versions achieved comparable and good results. The native implementations were chosen by most users for future use with 4 votes for the Android version and 5 votes for the iOS version, as shown in Figure 4. Figure 4 provides a graph overview of users preference for future usage of the tested versions.

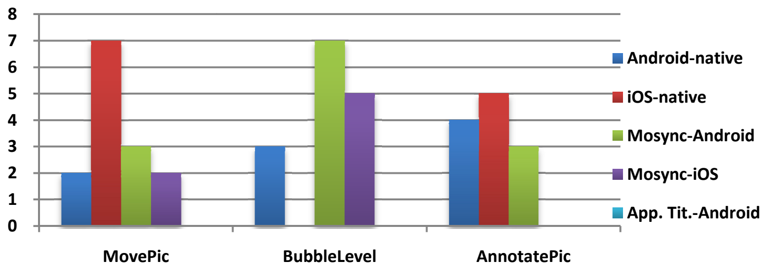


Fig. 4. Users preference for future usage of the tested versions

Overall, we analyzed that in case of those scenarios where the interaction-response time was very critical, i.e. in MovePic and BubbleLevel scenarios, users liked more the native versions because they received more quick response, but even so the cross-platform versions also gained good results. While in the case of scenarios where the interaction-response time was not so critical (i.e. in AnnotatePic scenario), all the versions received approximately the same kind of user satisfaction. Hence; we can conclude that where the quick response time is not so important, the better option is to use the cross-platform development as it saves time and cost and at the end achieves nearly the same user satisfaction level. While in the cases where the interaction response time plays an important role, if the quick response is not very critical then cross-platform development could be the alternative option; otherwise, the native-platforms are slightly better than the cross-platform frameworks. This is because the native implementation environments provide better solutions for the critical interaction response, which enhances the user satisfaction level. Overall, we can conclude that the mobile applications developed using the cross-platform frameworks provide approximately the same user satisfaction level as the ones developed in the native environments.

5 The Software Evaluation

We also performed evaluation of the developed versions of the three scenarios against software quality measurements to check the required development time between different platforms and frameworks. Moreover, we estimated their conformity against the style guides.

In the MovePic scenario, the native Android and iOS versions performed better. However, the image was following the fingers interaction slightly better in iOS. While the MoSync version's performance was the worst one due to the transformation between the 2D screen and the 3D OpenGL coordinate system. In the BubbleLevel scenario, the iOS version had the worst performance followed by the Appcelerator version. The reason behind Appcelerator version's low performance was the drawing of circle through a set of circular lines by the provided function, which seems to be not so efficient. While in the case of native iOS version, it was due to the technical difficulty occurred in the implementation for getting the quick request rate of the accelerometer data. The MoSync version had a far better performance but the best one was given by the native Android due to using a separate thread for the drawing. The AnnotatePic is not an interaction-response time critical scenario; hence, all the developed versions provided approximately the same performance level.

The implementations of the MovePic had a slightly different functionality for each version. The functionality provided by the native iOS version was better compared to the others, as it had the most natural interaction with the image and it also prevented the image from moving out of the screen, followed by the native Android version. In the BubbleLevel scenario, all implementations provided the same level of functionality due to the limitation of the scenario. In the AnnotatePic scenario, only the native Android version was without any errors while the iOS native version performed well until we updated it to the iOS 5.1. The MoSync version had problems during saving the images and was showing just half of the screen for the camera view.

From the maintainability perspective, the iOS-based versions were better compared to Android-based versions as much of the functionality was provided by the platform; hence, it shortens the code and reduces the time to develop. But as it is just for one platform only, so we felt the cross-platform development as the better solution. We also noticed that working with RhoMobile Rhodes framework saved development time compared to other frameworks as it provides many facilities for data-driven applications built-in from the start, followed by the iOS as it also provides much built-in functionality for utilizing many device features. In the case of cross-platform, we noticed that implementation time on MoSync was little bit faster than the others.

Lastly from the style-guide perspective, most of the cross-platform frameworks had not adopted the Android style guide. It is because the Android released its style-guide just few months before the time of our implementations. Moreover, we considered only the Android 2.2 and 2.3 versions during the development while some of the style-guide elements, like the action bar, work only on and above the Android 3.0. With regard to iOS based devices, we found out that all the tested cross-platform development frameworks provided pretty good adaptions to the iOS environment. Results of most of these frameworks conformed the style-guide of the iOS efficiently.

6 Conclusion

In this study, we performed a comparison between the native development environments and the cross-platform development environments. Apps were developed against three scenarios using the Android and iOS and native development environments as well as three selected cross-platform development environments (i.e. MoSync, Appcelerator Titanium, and RhoMobile Rhodes). The evaluation results from the software perspective and from the user study show that in many terms the results of cross-platform frameworks are as good as the native ones and in some cases even better. But the striking of Appcelerator Titanium has shown that relying on only one cross-platform development framework may lead to failures, because the whole smart-device market is evolving pretty fast and is in a constant flow. Also, when developing using the cross-platform approach, it is always better to think twice before updating an SDK to the newest version. It could be possible that the underlying cross-platform framework may not be able to build anymore on the new SDK version, as the developers of these frameworks need time for the adaption of the new SDKs.

The hybrid cross-platform frameworks provide much functionality today and it is surely further emerging. They also allow basic adaption and scalability to the tablets. The main difficulty for developers, who want to build cross-platform applications, is to find the solution best fitting their needs. But in most cases, it is not easy to find out what functionality a particular framework provides better than the others. In the forums of these frameworks many questions, which were posted one year ago or longer for asking some functionalities, are still looking for the answers or the answers are "coming soon functionality".

Overall, it can be said that the hybrid cross-platform frameworks are a good alternative to the native implementations with definite better cost-efficiency. But before choosing a particular framework, it is a must to find out if the underlying framework supports the needed features in a stable way. Moreover, the possibilities for porting existing applications between Android and iOS automatically are not fully developed, yet. The manual porting has some issues to remind but when considering them from the start, the process is manageable. In the near future, web cross-platform frameworks may compete the hybrid frameworks more and more, because of the fact that HTML5 is already capable of a few hardware access features and will perhaps evolve to replace the hybrid frameworks partially or totally. So, keeping eye on both technologies will help in deciding the better options for the underlying scenario.

References

1. A Look Back in Time at the First Smartphone Ever Business 2 Community, http://www.business2community.com/mobile-apps/a-look-back-in-time-at-the-first-smartphone-ever-040906
2. Di Giovanni, P., Romano, M., Sebillo, M., Tortora, G., Vitiello, G., Ginige, T., De Silva, L., Goonethilaka, J., Wikramanayake, G., Ginige, A.: User centered scenario based approach for developing mobile interfaces for Social Life Networks. In: UsARE 2012, pp. 18–24 (2012)
3. Gartner Says Android to Become No. 2 Worldwide Mobile Operating System in 2010 and Challenge Symbian for No. 1 Position by 2014, http://www.gartner.com/it/page.jsp?id=1434613 (last accessed February 21, 2013)
4. Heitkötter, H., Hanschke, S., Majchrzak, T.A.: Evaluating Cross-Platform Development Approaches for Mobile Applications. In: Cordeiro, J., Krempels, K.-H. (eds.) WEBIST 2012. LNBIP, vol. 140, pp. 120–138. Springer, Heidelberg (2013)
5. http://webinos.org/crossplatformtools/appcelerator-titanium/
6. http://webinos.org/crossplatformtools/rhomobile-motorola/
7. http://webinos.org/crossplatformtools/mosync/
8. Jones, S., Voskoglou, C., Vakulenko, M., Measom, V., Constantinou, A., Kapetanakis, M.: VisionMobile Cross-Platform Developer Tools (2012), http://www.visionmobile.com/blog/2012/02/crossplatformtools/
9. Paananen, T.: Smartphone Cross-Platform Frameworks A case study. Bachelor's Thesis. Jyväskylän Ammattikorkeakoulu - JAMK University of Applied Sciences (2011)
10. Palmieri, M., Singh, I., Cicchetti, A.: Comparison of cross-platform mobile development tools. In: ICIN 2012, October 8-11, pp. 179–186 (2012)
11. Smartphones setzen Siegeszug fort - Kalenderwoche 07 - 17. Februar - aetka Partnerforum (February 17, 2012), http://partnerforum.aetka.de/index.php?page=Thread&postID=993