User manual: hx-gameplay

1. Installation

To install hx-gameplay, use:

```
haxelib install hx-gameplay
```

Immediately after installing, run the post-installation script:

```
haxelib run hx-gameplay install
```

To use hx-gameplay, you also need another major Haxe library called hxcpp. However, always use the latest hxcpp version directly from the repository. To do so, first install hxcpp with `haxelib`:

```
haxelib install hxcpp
```

Then download the hxcpp repository with SVN or GIT:

```
svn checkout http://hxcpp.googlecode.com/svn/trunk/ hxcpp
git clone git://github.com/haxe-mirrors/hxcpp.git hxcpp
```

Then tell `haxelib` to use the downloaded repository instead of the default installation:

```
haxelib dev hxcpp absolute_path_to_the_hxcpp_repository
```

While you are at it, download and install the latest Haxe nightlies as outlined under the instructions on the Haxe homepage: http://haxe.org/download/manual_install#manual-install. As a word of advice, **always** use the latest Haxe nightlies, as Haxe is being developed quite dynamically. Don't fall behind. This concludes the installation of hx-gameplay.

The requirements for each of the supported target platforms are given in a later section.

2. Usage

There are currently 2 methods to use hx-gameplay: through a FlashDevelop project or through the command line. Templates for more IDEs are currently in production. Please note that there is no difference at all between FlashDevelop and command line projects. They are fully interchangeable. In fact, every command line project can also be opened in FlashDevelop. This allows you to easily work across different desktop platforms which is good for large team.

FlashDevelop

To use hx-gameplay with FlashDevelop, install the FlashDevelop template by doing:

```
haxelib run hx-gameplay install-flashdevelop-template
```

Now when you start FlashDevelop, you'll see the hx-gameplay template and you can use it to create projects. You can compile and test your application easily. Per default, every project includes the

windows-x86 platform and FlashDevelop is configured to execute it. If you need a different platform, first go to the project's directory on the command line and enter:

```
haxelib run hx-gameplay add-platform platform
```

Then open the project properties in FlashDevelop, in the "Test Project" section click "Edit…" and change the line:

```
exec windows-x86
```

to:

```
exec platform
```

You can also specify the release mode by specifying "release" or "debug" to the execution script. By default, the project is compiled in release mode. On some platforms, you must specify the build type and, depending on the platform, other arguments as well. For instance, when you build for Android you have to specify the release mode as well as the parameters to Apache Ant:

```
exec android-arm release debug install run
```

In this case, the C++ code will be compiled in release mode and Ant will generate a debug APK file for Android. Other platforms might use other options.

It is recommended to keep the project options in FlashDevelop synchronized with the Build.hxml file in the project's directory.

The next chapter discusses the project structure in more detail. Keep in mind that there's not really a difference between FlashDevelop and command line projects.

Command line

You can use hx-gameplay from the command line on all supported desktop platforms. The instructions here apply everywhere. To create a new hx-gameplay project from the command line execute:

```
haxelib run hx-gameplay create-project my-project-name
```

This creates a new directory for the project. Enter this directory. To compile the project enter:

```
haxe Build.hxml
```

Per default, every project comes with a suitable platform for the OS you are currently on. On Windows you'll get the windows-x86 platform, on Linux it will be linux-x86. After compiling the project, you can build and run it via:

```
exec platform {debug|release} {args}
./exec.sh platform {debug|release} {args}
```

If you wish to build and run the project separately you can use the scripts:

```
platforms\platform\build.bat {arguments-to-hxcpp}
```

```
platforms/platform/build.sh {arguments-to-hxcpp}
run-platform.bat {debug|release} {args}
./run-platform.sh {debug|release} {args}
```

The meaning of the files – and directories in the project is as follows:

- **platforms/** – holds the run-time files for the supported platforms as well as the build files in the hidden .obj directory.
- **platforms/platform** – holds the run-time files as well as the build script for the specific platform.
- **platform/build.bat, platform/build.sh** – the build script for the specific platform, by invoking from the main project directory you build the project for this platform. If you pass arguments, they will be delegated to hxcpp.
- **platforms/bin** – holds run-time files as well as additional platform-specific files.
- **platforms/.obj** – holds the build files for the supported platforms. The build files are cached, so you don't have to rebuild the whole project for each every change. However, build files for modules you no longer need are not automatically deleted. If you ever come across some non-understandable build error, try to delete the build directory for the platform.
- **res/** – holds all of the application's resources. This directory is copied to platforms/platform/bin by the respective run script.
- **game.config** – holds the Gameplay application's configuration. This file is copied to platforms/platforms/bin by the respective run script.
- **src/** – per default holds the application's source code. Additional source paths can be easily added.
- **.cpp/** – contains the C++ code generate from Haxe. This is the same for all platforms.
- **Build.hxml** – invoke `haxe Build.hxml` to compile the project.
- **Project.hxproj** – the FlashDevelop projct file for this project.
- **exec.bat, exec.sh** – shortcuts allowing you to build and run the project quickly. Especially useful in FlashDevelop to run the project after compilation.
- **run-platform.bat, run-platform.sh** – these scripts provide an easy way to start the application quickly.

As you see the project structure is highly modular. You add, remove, build and run the project for multiple platforms easily. Also, the system is designed to report every possible error to you so you can resolve it quickly.

3. Requirements

This section lists what is required to build and run Gameplay projects for the individual supported platforms.

Windows

To build on Windows you need the Microsoft Visual Studio 2010. The Express version is fine as well. It has to be the 2010 edition, later and earlier versions do not work. Although hxcpp supports GCC on

Windows as well, this setup is currently not supported. Make sure the environment variable VS100COMMONTOOLS is set after installation and hxcpp can find the Visual Studio compiler.

On Windows XP, you'll need the robocopy.exe tool by Microsoft. You can get it by quickly installing this package http://www.microsoft.com/en-us/download/details.aspx?id=17657.

Linux

To build on Linux you need the GCC compiler suite mostly. If hxcpp complains for missing libraries, you can install them with tools like apt-get. On Ubuntu, you probably must only install g++ manually.

Android

To build for Android you require the following things:

- **Android SDK** – install it and set the ANDROID_HOME environment variable to point to it, e.g. C:\android-sdk.
- **Android NDK v.7** – hx-gameplay has been tested with the v.7, but there are good chances that higher versions will work as well. If you really need to use another NDK, you must edit the build scripts as well. Set the ANDROID_NDK_ROOT environment variable to point to it, e.g. C:\android-ndk-v7.
- **Apache Ant** – download it and set the ANT_HOME environment variable to point to it, .e.g. C:\apache-ant

If you experience problems, such as hxcpp ignoring the environment variables, search for the hxcpp config file .hxcpp_config.xml in your home directory. It might be overriding the variables with different values. Delete the entries there and use the environment variables if this is the case. If you are on Linux or another Unix derivate, a good place to set environment variables is /etc/profile.

Note: the Android build process is delegated to Apache Ant. Because of that you must always specify the desired build mode as well as the desired packaging mode to the automation scripts:

```
exec.bat release debug install
run-android-arm.bat release debug install
```

The first argument is always the build mode specifier.

4. API

The structure of the API corresponds very closely to that of the C++ library as described here on the homepage of the library: http://gameplay3d.org/api.php. Class names are the same, inner classes are named ParentClassName_ChildClassName, same for nested enums nd structures. Because there's no official for the API yet, consult the samples that come with hx-gameplay.

5. Samples

Included in hx-gameplay are the standard samples provided by the Gameplay development team. They are normal command-line projects and you can easily open them in FlashDevelop or build them from the command-line as described in the previous chapters. Per default, the samples include no target platforms.

6. Troubleshooting

If you have issues you can't fix or if you get error messages you don't understand, there are 3 main places to get help:

1. The Haxe forum at http://haxe.org/forum;
2. The Haxe Google Group https://groups.google.com/forum/#!forum/haxelang;
3. The Haxe IRC channel #haxe on Freenode.

You can also get in touch with the creators of hx-gameplay easily by mail or Skype. Just look or ask around for some contact data.