AppAble for Flash Applications

Using the Zinc Excecutable Wrapper

Introduction:

The baKno AppAble allows Flash developers to make desktop applications that connects to Intel's app store using the Zinc executable Wrapper. Zinc allows developers to do much more with the applications with the help of extensions that enables them to do things like access databases or call libraries from the operating system.

Package Contents:

- The Software License Agreement (License.txt)
- The AppAble External files (Externals)
- The dependency's libraries (Microsoft.VC90.CRT)
- A sample Flash/Zinc project that connects to the store
- A guide to create the MSI file for submission

Developer requirements:

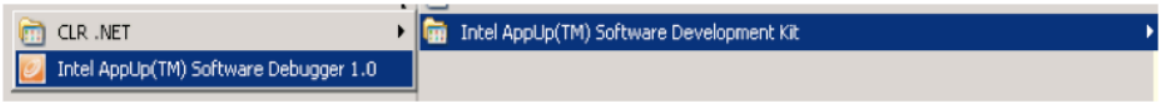
- The Intel's App store SDK. It can be obtained here: http://appdeveloper.intel.com/en-us/
- Microsoft's runtime libraries (provided with the package).
- Adobe Flash with ActionScript 2 or 3.

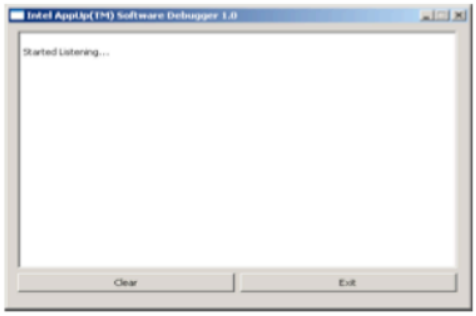
- MDM Zinc™ 3.0 Buider available here http://www.multidmedia.com/software/zinc/

How to use the SDK:

There are two environments, Debug and Production. The idea is to develop under Debug and only when everything works, change to Production to build the final application and submit it to the AppUp store.

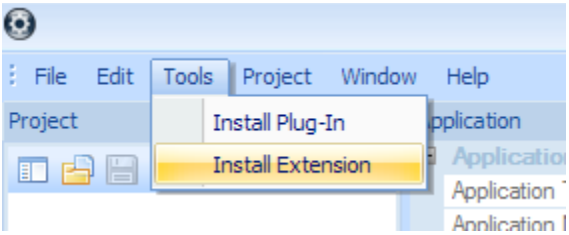
To test your Debug application you need to run the “Intel AppUp Software Debugger”, an application that emulates a communication to the AppUp store. Then run the Debug application and check if all communication stages work as described on the docs. This Debugger is run by calling it from the Start Menu and going to the Intel SDK program folder.



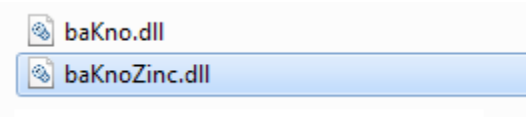


Add the Extension to Zinc:

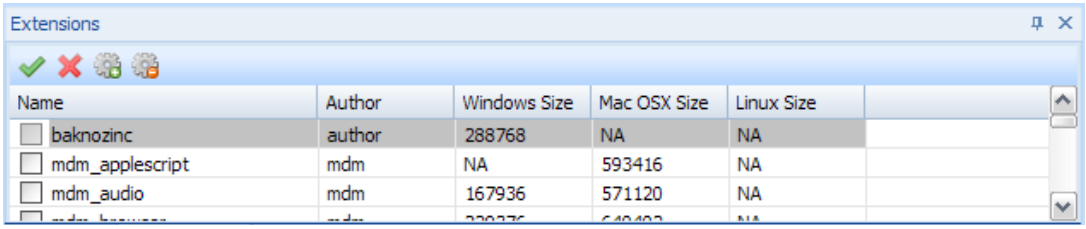
After MDM Zinc™ 3.0 is installed we need to add the Dll Extension. We open the environment and on the menu we select the option “Tools->Install Extension”



And explore for the Dll extension “baKnoZinc.dll” within the External directory. The instalation is done automatically .



If successful, the extension will be seen in the extensions tab inside the Zinc Environment

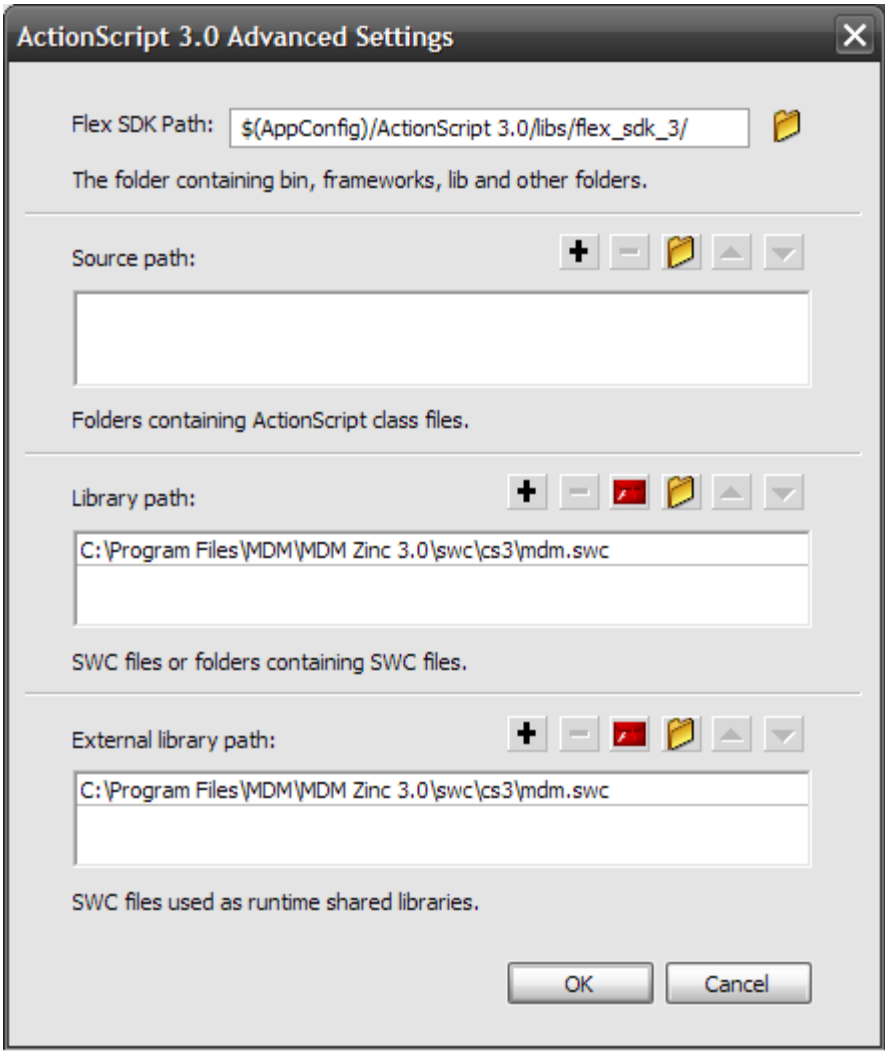


Adding the Zinc SWC File to the Libraries for ActionScript

In order to publish your Flash application, we need a reference to the SWC file that Zinc provides, mdm.swc. In a default Zinc installation the path should be

C:\Program Files\MDM\MDM Zinc 3.0\swc\cs3\mdm.swc

In order to add the reference, in Flash go to Edit->Preferences, then on the ActionScript tab go to the ActionScript 3 settings and add the path for the library as the following image shows.

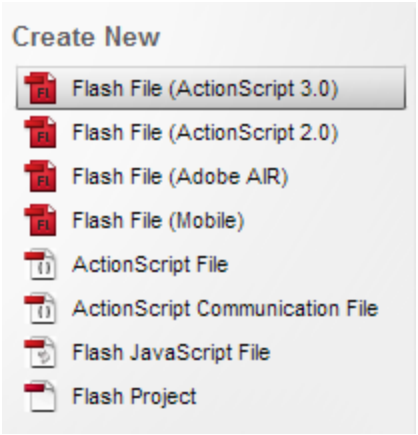


Developing a Simple Flash Application:

The example application does the following steps:

- Connects to the Client Agent
- Checks the Authorization status
- Begins an event
- Ends the event
- Disconnects from the Client Agent
- Displays all messages in text boxes

First we need to create a simple flash file with ActionScript 3 (AS2 will be discussed later on this document).



On the empty space we add 5 buttons and 5 text boxes, one next to a button.

Rename the buttons as “Connect”, “Status”, “BeginEvent”, “EndEvent” and “Disconnect”.

- For connection, we give the button the instance name ***btConnect*** and the text box ***txConnect.***
- For status, we give the button the instance name ***btStatus*** and the text box ***txStatus.***
- *For beginning the event, we give the button the instance name* **btBeginEvent** and the text box ***txBegin.***
- For ending the even, we give the button the instance name ***btEndEvent*** and the text box ***txEnd.***

- *For disconnection, we give the button the instance name **btDisconnect*** and the text box ***txDisconnect***

Then we go to the timeline and on the first frame we add the script (F9 is the shortcut)

The first line is to import the Zinc libraries into the script

```
import mdm.*
```

we must also initialize the Zinc application in order to use the libraries

```
mdm.Application.init(this, onInit);
function onInit() {
  //Place code here
}
```

mdm.Application.init Indicates to initializeZinc and the return event should be handled by the ***onInit*** function

We then proceed to add the connection event when the Connect button is pressed

```
btConnect.addEventListener("click", Connect);
function Connect(e) {
        var myReturn = mdm.Extensions.getExtensionByName("baKno").connectToIntelADP("0x11111111",
                "0x11111111", "0x11111111", "0x11111111",
                "11111111-1111-1111-1111-111111111111")
 txConnect.text = GetMessage(myReturn)
}
```

the GetMessage function that is in the script allows to give a string message that corresponds to the returned value from the Client Agent.

We add the status button event.

```
 btStatus.addEventListener("click", Status);
function Status(e) {
        var myReturn = mdm.Extensions.getExtensionByName("baKno").checkAuthorizationStatus()
   txStatus.text = GetMessage(myReturn)
}
```

Sometimes is is required to record use time. For that reason, two commands are present to do the job easily. Remember that “every beginning has an end”, it is recommended to use these two functions in pair. In the example, each have its own event.

```
btBeginEvent.addEventListener("click", BeginEvent);
function BeginEvent(e) {
```

```
        var myReturn = mdm.Extensions.getExtensionByName("baKno").beginEvent()
    txBegin.text = GetMessage(myReturn)
}

btEndEvent.addEventListener("click", EndEvent);
function EndEvent(e) {
        var myReturn = mdm.Extensions.getExtensionByName("baKno").endEvent()
    txEnd.text = GetMessage(myReturn)
}
```

And to disconnect, the following event is needed

```
btDisconnect.addEventListener("click", Disconnect);
function Disconnect(e) {
        var myReturn = mdm.Extensions.getExtensionByName("baKno").disconnectToIntelADP()
    txDisconnect.text = GetMessage(myReturn)
}
```

The Intel AppUp SDK also offers an upgrade feature, to get other applications as well, this is present in our wrapper as well with the following function (Optional):

```
btUpgrade.addEventListener("click", Disconnect);
function Upgrade(e) {
        var myReturn = mdm.Extensions.getExtensionByName("baKno").upgrade("0x22222222",
                        "0x33333333", "0x44444444", "0x55555555")
    txUpgrade.text = GetMessage(myReturn)
}
```

The full source of the scrip can be found in the sample flash file and a separate text file that are included in the package.

A Note About Debugging:

With the Zinc libraries, it's possible that the debugging of the application does not work at all. It's an issue with Zinc and Flash and not the package.

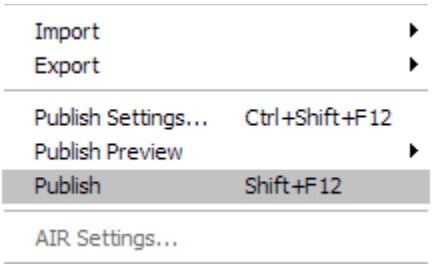
A note about ActionScript 2:

The events are the same but the syntax is different, here is a sample for the status event

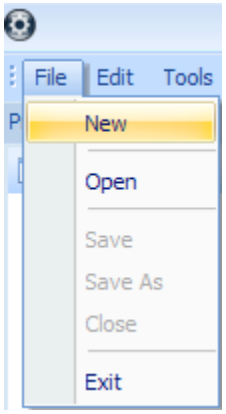
```
btStatus.onRelease = function() {
        var myReturn = mdm.Extensions.baKno.checkAuthorizationStatus ()
        txStatus.text = myReturn
}
```

Deployment of the Application:

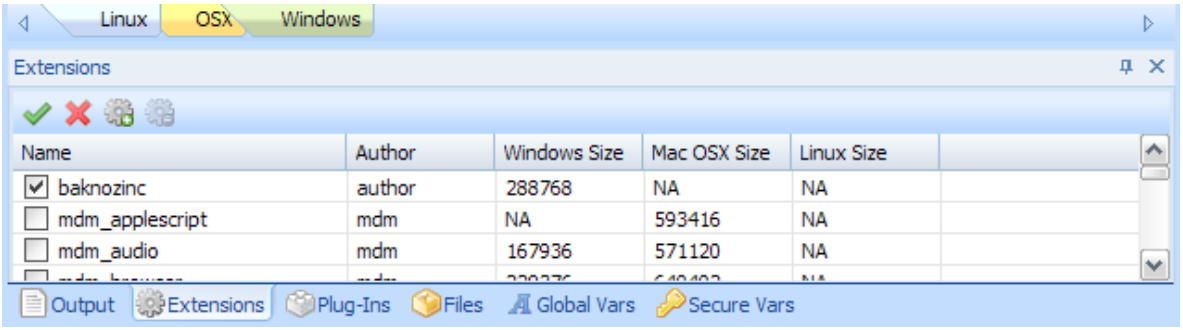
Once the code is complete, the flash file must be published as a swf file. This can be done by exporting or by the publish option in the File menu.



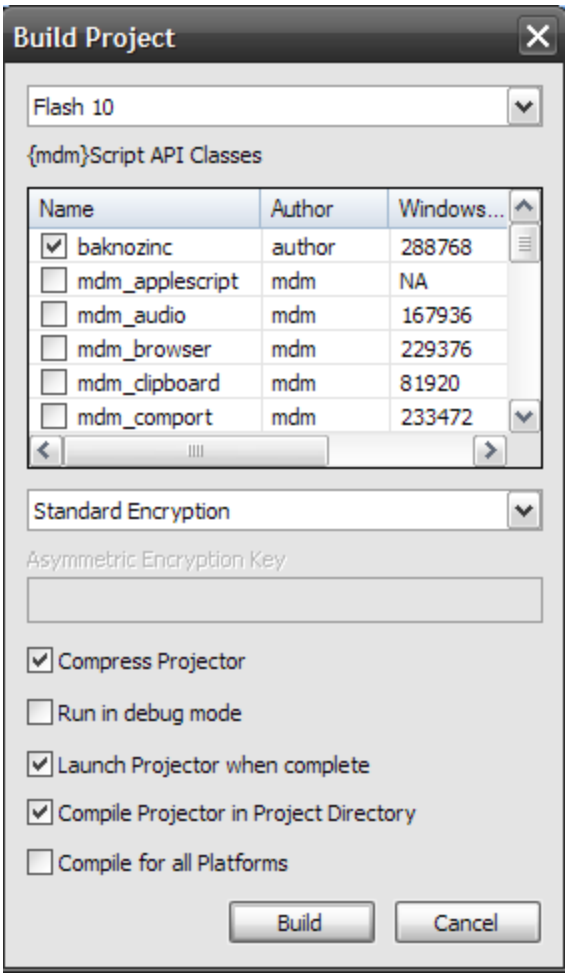
Once this is done, we close Flash and open Zinc. Then we add the flash file by going to File->New in the menu.



Choose the flash SWF file created earlier and the system will import it instantly. Before we wrap the flash file as an exe application, we must add the baknozinc extension so that the libraries are used.

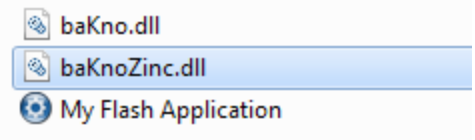


To deploy the exe file we go to Project->Build project in the Menu, we make sure the baknozinc extention is selected and hit “Build”



With the default options the application will execute right after it's built. So it's ready to test. Just start the ADTS provided by Intel SDK and try each of the buttons. Make sure

before that that both baKno.Dll and baKnoZinc.dll are on the same directory as the application



To debug the applicatiuon in your development computer, the Debugger from the SDK must start before the demo is executed.

List of Commands:

1. **Connect to the App store:**

   connectToIntelADP(string l1, string l2, string l3, string l4, string baKnoID) :int

   This command initializes the connection to the App store. Each of the four first parameters are part of the GUID that the Intel developer program provides to the developer. The last parameter is the GUID that baKno provides to clients that enables its use.

   Note: Because of the current development status of the SDK, The parameter values must be in hexadecimal of size 8 exactly (i.e.:0x12345678).

   So, if the id provided is: 12341234-2345-3456-4567-567890123456

   The parameters are: 0x12341234, 0x23453456, 0x45675678 and 0x90123456

   Note: For developer purposes, the SDK provides a debugging ID so developers can test their applications. Each parameter has the value 0x11111111. baKno also provides a debug GUID. Thus, the command to connect to the developer's App store is:

   var a = connectToIntelADP("0x11111111", "0x11111111", "0x11111111","0x11111111",

   "11111111-1111-1111-1111-111111111111");

   Return values:

   - **-1:** The Application presented an Unknown Error.
   - **-2:** The file**,** baKno.Dll, is invalid.
   - **-3:** Could not locate the file baKno.Dll.

- **-4:** The input parameters are invalid.
- **-5:** There are not 4 parameters in the function call.
- **-10** The baKnoID is invalid.
- **0:** The Application started successfully.
- **1:** The application could not start.
- **2:** The Intel's Client Agent is not Available.
- **6:** The Application is not authorized.
- **7:** The Application ID has expired.
- **9:** The connection with the Client Agent reached a timeout.

2. **Check the Application Status:**

   checkAuthorizationStatus():int

   Command that checks the current status of the Application

   Return Values:

   - **5:** The Application is authorized
   - **6:** The Application is unauthorized.
   - **7:** The Application ID has expired.
   - **9:** The connection with the Client Agent reached a timeout**.**

3. **Save The Application Runtime (Optional)**

   beginEvent():int

   Command that allows the App store client to start recording the Application's runtime.

   EndEvent():int

   Command that ends the recording of the Application's runtime.

   These two commands allow the developer to obtain statistics about the time each customer runs the Application.

Note: Both commands MUST be used together, meaning, that with each time *beginEvent* is used, there must be *EndEvent* command later on.

Return Values:

- **-1**: The Application presented an unknown error.
- **0:** The command has been executed successfully.
- **6:** The Application is not authorized.
- **7:** The Application ID has expired
- **8:** The "begin event" has not been called.
- **9:** The connection with the Client Agent reached a timeout.

4. **End the Connection:**

disconnectToIntelADP():int

Command that terminates the connection to Intel's App store.

Note: This command MUST be executed when the end uses closes the Application. If it's executed earlier, the application may close by itself immediately. This is due that the App store has a record of the application that is running and terminates the process automatically.

5. **Upgrade:**

upgrade(string l1, string l2, string l3, string l4):int

This function calls the upgrade to the next application. The four string parameters represent the new GUID to call.

Return values:

- **-1:** The Application presented an Unknown Error.
- **0:** The process started successfully.
- **1:** The process could not start.
- **6:** The Application is not authorized.
- **7:** The Application ID has expired.