This page last changed on Apr 27, 2009 by beechc.

AGtoolkit API: Introduction -

Welcome to the AGtoolkit SDK Flash documentation. The AGtoolkit is designed as an interface between your Game and AddicitingGames HighScore system and the social Profiles database. It allows you to construct your Game to take advantage of these features and create a richer experience for Users, by utilizing the tools packaged in the AGtoolkit. At this writing, the AGtoolkit is in it's early development stages, and in the future will provide greater access to many aspects of the social network as we continue to expand it's capabilities.

The AGtoolkit is designed to be easy to install and use, consisting of a single Flash Component, and API methods that allow it's functionality. The AGtoolkit Component for both AS2 and AS3 Flash langauges contain all of the same methods and implementations described in this document. In Flash CS3 and later you may install both AS2 and AS3 components and Flash will present the appropriate component language version in your domument's Component panel.

AGtoolkit API: Installation -

Your AGtoolkit Component is simple to install using the Adobe Extension Manager. Simply double-click on the Component installation .mxp file and follow the normal Component installation procedures. The AGtoolkit will then be available to you within the Standard Components directory of the Components panel in the Flash IDE, if Flash was open during installation, either reboot Flash or use the Components panel 'reload' command to refresh the panel.

To make the AGtoolkit available to your Game, drag an instance of the AGtoolkit Component to your Game's **Library** - DO NOT place the AGtoolkit Component on the Stage, it only needs to reside within the documents's Library panel in order to function, and will cause Flash Compiler errors on publish if placed on the Stage directly. Now you are ready to use the AGtoolkit to link your game to the AddictingGames HighScore game social network.

AGtoolkit version upgrades:

Use the following installation procedure only if you have previously installed the AGtoolkit Component and are installing an upgrade version:

1) launch the Adobe Extension Manager, select the currently installed AGtoolkit Component, delete using the 'trash' icon

2) double click on the new AGtoolkit upgrade version .mxp file, and install the component

3) launch Flash, open the game document containing the previous AGtoolkit version, delete the AGtoolkit Component from the Library

4) drag a new instance of the AGtoolkit Component to the file Library, save the file.

AGtoolkit API: Contents -

The AGtoolkit consists of the AGtoolkit API, the AGtoolbar and several AGscreens creating a set of UIs that allow access to common User functions. It monitors and detects User login status, embed location and play statistics. It provides Sharing controls, in-game AddictingGames teasers and advertising, User and Friend Data, and score submission to the AddictingGames HighScore database, all within the background of your games normal operation. Additionally, the AGtoolkit provides you with several features that can make building your game easier.

The AGtoolbar provides a set of common UI controls including restart, pause, sharing, and volume controls. The AGtoolbar global volume control system is designed to contol all game sound, you do not need to do anything in your Game code to cause this operation, and no longer need to build these elements into your system. However the AGtoolkit does provide access to these elements in the event that your Game requires specialized Sound controls.

The AGtoolkit provides a 'game over' screen, as well as several other screens for elements such as User login, Score submission prompts, Leaderboards, and User notifications. These screens are automatically implemented at the appropriate times, however you will invoke their sequence at endgame.

to implement the AGtoolkit, there are a few requirements that you must follow:
- Your Game must have minimum dimensions of 400x340, and be no greater than 800x640 (heights include the AGtoolbar area)
- Your Game must extend the Stage Height by 40 pixels beyond your game's viewing area, to accomidate the placement of the AGtoolbar UI - Alternatively, you do not have to extend the Stage Height, however AGtoolbar UI will overlay the bottom 40 pixels of the Stage
- Your Game must implement the AGtoolkit API methods in the order described herein, as the first operation in your game system's code construction

The AGtoolkit must load external assets into your Game at runtime in order to provide a dynamically flexible system, in addition there are several procedures and routines that must run at the onset of Game instantiation to provide the necessary communications protocols with AddictingGames servers. This means it is imparative that you follow these guildelines in order to succesfully integrate your Game with the needs of the AGtoolkit as a trusted AddictingGames developer.

The AGtoolbar will overlay the gameface at the bottom of the Stage using a 40 pixel high area that will dynamically extend to fit the width of your Stage. We understand the complications that this can create for your code, however at this time there is no alternative to this UI requirement. You will need to accomidate the change in Stage Height within you game code calculations, if in the event it effects your system in some way. For example, if your code uses - "stage.stageHeight / 2" - as a centering operation, your calculation may not function as expected since the Stage is being extended (or overlays), and the visual 'game window' is 40 pixels less. A similar situation may arise for Stage 'bounds' checks using stage.stageHeight as the maximum y parameter - for instance, a bouncing ball. Please adjust your code or program accordingly to account for this necessity, in most situations using - (stage.stageHeight - 40) - will be sufficient to make the needed calculation.

Make certain that you only instantiate the AGtoolkit constructor once within your code. If you are using a timeline based system, be sure that you do not loop back to frame one (or the constructor's instantiation frame) as this will cause the constructor to attempt reinstantiation.

AGtoolkit: Constructor -

new AGtoolkit(scope:Object, gameID:String, gameTitle:String, gameAuthor:String):void

parameters	description
scope:Object	simple reference to the root document timeline, in almost all cases it should be the standard keyword **this**
gameID:String	the game's referencing identification number, as a String, assigned to the game by your Producer

gameTitle:String	the game's title, as a String, assigned to your game by your Producer
gameAuthor:String	you or your Company

The AGtoolkit constructor statement must be instantiated as one of the first declarations in your game code, as the AGtoolkit needs to load the required assets, and perform critical pregame tasks and settings. As an AddictingGames developer, your Producer will assign a gameID and gameTitle for your game. You may declare the constructor statement either using timeline code on the first frame, or within the constructor statement of your Document class file. When declaring the constructor, assign it to a local property reference so that you may easily target methods of the AGtoolkit in later areas of your game code. A typical AGtoolkit constructor statement will appear similar to the following:

timeline based constructor:

```
var myToolkit:AGtoolkit = new AGtoolkit(this, '1234', 'myCoolGame', 'Cool Game Studios');
```

class based constructor:

```
public var myToolkit:AGtoolkit;
public function myGameDocumentClass():void {
        myToolkit = new AGtoolkit(this, '1234', 'myCoolGame', 'Cool Game Studios');
}
```

Note: You may use any property name you wish to reference the AGtoolkit within your game code (ie. "myToolkit")

AGtoolkit: AGtoolbar -

The AGtoolbar contains a variety of functions and controls, some of which are static, others that may or may not be required by your game or by your Producer. There are three static elements of the AGtoolbar: "gigya", "mute", and "volume". These will always appear on the far right of the AGtoolbar. There are two optional control buttons of the AGtoolbar: "restart", and "pause". These controls will only appear in the AGtoolbar if instantiated via the AGsetToolbar() method. In most games, we will require the use of the "restart" button, however we also understand that some games will not have need of this element depending on the genre and style of your game. Consult your Producer to determine the needs and requirements for your game system

When assigning a AGtoolbar button control to your game, you are required to create a callback method that will be fired by the AGtoolbar button. This is designed so that you may then perform the tasks as needed by your game system to produce the action required by the button. In addition, the "mute" and "volume" controls also may be assigned callback methods, if your game requires special Sound control operations in the event that one of these buttons are triggered by a user. Although these buttons are static and control all Sounds globally (with the exception of loaded or streamed Sounds), this is designed to give you complete control over your games Sound features, if required.

Each of the AGtoobar callback methods, with the exception of the "restart" button, will return a value when triggered that is related to the type of action taken. Although you are not required to use this value, you must account for the parameter within your callback methods declaration statement. This is designed as a conveince value that you may use to perform special actions without having to track these values within your own game code.

All AGtoolbar assignments (eg. the use of the AGsetToolbar method) should be made previous to calling the AGinitToolkit() game initiation method.

method: AGsetToolbar(element:String, callback:Function):void

parameters	description
element:String	the name of the AGtoolbar element the call is targeting
callback:Function	your Game code method that will be invoked on the triggering of the AGtoolbar element

The AGsetToolbar method allows you to link the AGtoolbar controls to methods of your code so that they will interact with your Game. The four exposed AGtoolbar element names are: "restart", "pause", "mute", and "volume". Use one of these four values within the 'element' parameter of the AGsetToolbar method when assigning the control to your system. You must also create a callback method that correspondes to the assignment made within the 'callback' parameter. Within your callback method declaration you may name the return parameter variable anything you wish, however you must use the correct datatype value according to the element being assigned, as listed below:

element value	return datatype	return value	return description
restart	void	n/a	n/a
pause	String	'on' || 'off'	'on' = running, 'off' = paused
mute	String	'on' || 'off'	'on' = playing, 'off' = muted
volume	Number	0 - 1	volume level percentage

You ONLY need to call the AGsetToolbar method IF you are using or assigning a method to one or more of the AGtoolbar elements, however for each element you must create a separate statement. The following demonstrates the use of all four element assignments, in addition to the callback declarations assigned to each AGtoolbar element (you can use any named method of your preference, as long as you assign it within the AGtoolbar method).

```
myToolkit.AGsetToolbar('restart', myGameRestart);
myToolkit.AGsetToolbar('pause', myGamePause);
myToolkit.AGsetToolbar('mute', myGameMute);
myToolkit.AGsetToolbar('volume', myGameVolume);

//declared methods in your game code
function myGameRestart():void {//your code goes here... }
function myGamePause(state:String):void {//your code goes here... }
function myGameMute(state:String):void {//your code goes here... }
function myGameVolume(volume:Number):void {//your code goes here... }
```

Note: you must preface all AGtoolkit API method calls with your local reference property (ie. "myToolkit")

method: AGpauseScreen(default:Boolean=true):void

The AGtoolkit contains a built in pause screen that will overlay the game stage when the AGtoolbar pause button is activated by a user. If you have implemented the AGtoolbar pause button control, and you wish to use your own game's custom pause screen, set the AGpauseScreen control parameter value to "false". However if you do so, you will need to implement your screen's acitvation/deactivation within your assigned "pause" button callback method.

```
myToolkit.AGpauseScreen(false);
```

AGtoolkit: AGgameScore -

The AGtoolkit is designed as a game interface for use with AddictingGames HighScore Leaderboards. As such, your 'main' game statistic (often 'score') will be used to determine the Rank of Users who play your game. When your game completes and the AGgameover() method is called, this statistic will be used in constructing and detemining several elements, and will be presented to the User during the game over process. However, our system is designed so that you do not need pass this value at endgame, instead simply 'registering' your established game code property with the AGtoolkit is all that is required, the AGtoolkit will retrieve the current value (at endgame) and process it accordingly.

method: AGgameScore(prop:String, label:String, type:String, precision:Number=0):void

parameter	description
prop:String	the name of the property member within your game code specified as a String. this statistic should always be datatyped as a Number in your code and must reside within document the root scope.
label:String	the linguistic description of your statistic (ie. 'points', 'kills', 'shots', etc.) presented to the user, following the statistic value
type:String	the statistic's value format (see below)
precision:Number	the statistic's decimal position accuracy, passed if the value is of types: float, percent or currency

The following demostrates the registration of your game code property with the AGtoolkit, your code's property name may be anything that you wish, provided that you assign the correct name (as a String) in the prop parameter of the AGgameScore method.

```
//property member declared in your game code
var score:Number = 0;

//AGtoolkit statistic registration
myToolkit.AGgameScore('score', 'points', 'integer');
```

AGgameScore value formats - type property:

The AGtoolkit currently supports the following value type formats that you must specify in order to display the value correctly. When declaring your AGgameScore properies you must pass the correct type property for the value of the statistic, this relates to AGtoolkit processing as well as database storage and retrival, it is necessary in order to present the statistic universally across several programming languages.

type	output format
integer	000,000,000
float	000,000,000.0000
time	00:00:00
percent	000.0000%
currency	$000,000,000.0000

integer:

Integer values represent a whole number and have a maxmum value of 999,999,999

float:

Float values represent a floating point decimal number. When using a float value type format you must also specify the 'precision' of the value or the number of decimal places used for display. The AGtoolkit currently supports 0 - 4 decimal precision and a value maximum of 999,999,999.9999

time:

Time values are to be tracked in milliseconds and will be parsed to display the minimum number of characters needed as: hours, minutes, and seconds.

percent:

Percentage values range from 0.0 - 1.0 and may present accuracy of up to 4 decimal precision. When using percentage based statistics be sure that your values remain under 1.0

currency:

Currency values may present accurancy of up to 4 decimal precision and have a maximum value of $999,999,999.9999 - the AGtoolkit currently only supports '$' amounts and formatting, future versions will support a wider range of currency types.

AGtoolkit: AGinitToolkit -

The AGinitToolkit is the final method you call in order to launch the initialization sequence and ultimately begin you game. Once you have set any needed properties of the AGtoolkit you wish to invoke, call this method and pass your game's init method as a parameter. When all AGtoolkit processes complete there cycles, your game init method will be fired to begin your processing sequence. You should call the AGinitToolkit method AFTER all other AGtoolkit property declarations have been made, in order to make certain that these elements are properly instantiated and operate correctly.

method: AGinitToolkit(gameInit:Function):void

parameters	description

gameInit:Function	this is your games initialization method as declared within your code

```
myToolkit.AGinitToolkit(myGameStart);

//your declared initilization method
function myGameStart():void {
  //your code goes here
}
```

AGtoolkit: AGgameover -

The AGgameover method initiates the game over sequence at the end of your games operation, call the AGgameover method when your gameplay completes. Make sure that you have stopped all operating processes and updated alll game statistic final values previous to calling the AGgameover method. When called, the AGtoolkit will present a sequence of 'game over' screens, this UI displays the results of the previous game (eg. the 'score') and prompts the User through a series of screens including, login and registration, score submission, notifications, and restart. The AGtoolkit makes all the determinations necessary and respondes to all of the required steps presented, until ultimately the game is restarted by the User, which then triggers the callback 'restart' method you have provided in your game code.

You must pass your game's declared 'restart' method as the gameRestart parameter, in order for the AGtoolkit to restart your game operations when invoked by a User. This method can be the 'same' callback method used to restart your game if you have invoked the AGtoolbar 'restart' button element, if so you do not need to pass the gameRestart parameter in the AGgameover method call.

method: AGgameover(gameRestart:Function):void

parameters	description
gameRestart:Function	this is your game's restart method as declared in your code

```
//your declared game over process
function myGameOverHandler():void {
  //your other code here ...
  myToolkit.AGgameover(myGameRestart);
}

//your declared restart method
function myGameRestart():void {
  //your code goes here ...
}
```

AGtookit: AGuserdata -

The AGtoolkit provides access to basic User Profile information that you can use within your game to enhace the experience. These methods return Objects containing data on the currently 'logged in' User. If the AGtoolkit detects that there is not a User currently logged in, or if the game is being played on a Domain other than AddictingGames Domains, the methods will return 'null'. You should always use a conditional statement to determine if the User is present and logged in before attempting to use data contained in the Object properties, if you do not Flash will throw a 1009 null object reference exception, and your script will terminate.

Unlike setting AGtoolkit properties previous to the AGtoolkit.AGinitToolkit() method, AGuserData calls should be made AFTER the initialization of the AGtoolkit, and usually within the game assigned initilization method that is called once initial AGtoolkit processes are complete. This ensures that all necessary communication systems are ready to handle the call processing and return the requested data.

Each **userObject** consists of standard Flash notation, name=value pairs. All userObjects will contain the following structure:

userObject:

property	description
name:String	the username of the current User
rank:Number	the current rank of the User for your game
score:Number	the current High Score of the User for your game
time:Number	the UTC timestamp of the last High Score
image:String	the URL of the current Users avatar image
page:String	the URL of the current Users profile page

Users:

To obtain the userObject of the currently logged in user, call the AGgetUser method. It is often advantageous to do so at the initialization of your game, as well as several othe points, for instance on restart. In this way you continue to confirm that the infomation is available for your game to use. If it is not, you must provide a means for you game to run without a User being logged in - eg. your game cannot ONLY function if a User is available, and must be prepared to handle an 'anonymous' player contingency.

When making a call on AGuserData methods, you must assign a callback method that will handle the returned value. The nature of these calls are asynchronous, meaning that the AGtoolkit must query the server, gather the needed data and return it to the game - this takes time. As a result the values are not 'instantly' availble to your game, so you must design a system structure that allows for this and steps through the initialization process to gather the data you intend to use during gameplay. Calls to AGuserData also invoke the AGloadingScreen which will overlay the game face until the return value is received.

method: AGgetUser(callback:Function):void

parameters	description
callback:Function	the method invoked when data is received

callback return:userObject

```
//your declared local property
var myUser:Object;

//your game init procedure
```

```
function initGame():void {
  //your other init codes here ...
  mytoolkit.AGgetUser(myUserReturn);
}

//your declared return handler
function myUserReturn(userObject:Object):void {
  myUser = userObject;
  startMyGame(); //example method to invoke after the return is complete
}
```

There are several ways to use the properties of the userObject within you game, once it has been loaded:

```
//use a condition statement to determine if the user is logged in before calling any property
if(myUser != null) {
  usernameTxt.text = myUser.name;
  userrankTxt.text = myUser.rank;
}

//load the user avatar
if(myUser != null) {
  var loader:Loader = new Loader();
  var request:URLRequest = new URLRequest();
  request.url = myUser.image;
  loader.load(request);
}

//create a link to a User profile page
if(myUser != null) {
  var link:URLRequest = new URLRequest(myUser.page);
  myLinkBtn.addEventListener(MouseEvent.CLICK, function(){navigateToURL(link, '_blank');});
}
```

Leaderboards:

The AGtoolkit is designed as a means to create community based HighScore games. Your game will use the AG HighScore system to submit User scores which are presented in Leaderboards on your game page, including categories for 'Everyone' and 'Friends', and time periods of 'AllTime' and 'Week'. The AGgetLeaderboard method will return as many as the top 100 positions, for either category and period passed in the parameters

The returned Leaderboard data will be in the form of an Array containing userObjects with the same properties as listed above. However this method, like the above, also returns 'null' if the current User is not logged in, and you must provide means for your game to run without the need for a Leaderboard to be present. From this data you can determine the User's current rank position among other Leaderboard members within your game during runtime, present the User with specific messaging, utilize User avatars as part of game play, or any number of other creative methods to make your game more exciting and interactive for the User and the AddictingGames community.

As with the above AGgetUser() method, the AGgetLeaderboard method is an asynchronous return, and requires a callback method pass within it's parameters. You should also check the data value for a null return before attempting to use data contained within the Leaderboard Object, as well as using an initilization procedure to ensure that the values have been retreived.

Remember that in all AGleaderboard returns, the current User is included by default. This means that you may wish to check the userObject you are utilizing in an operation, and filter it for the current User if this does not work for the system you are designing.

method: AGgetLeaderboard(callback:Function, group:String, period:String):void

parameters	accepted values	description
callback:Function	user defined	the method invoked when data is received
group:String	Everyone, Friends	the Leaderboard group type you are retrieving
period:String	AllTime, Week	the Leaderboard period type your are retrieving

callback return:userArray

```
/* extends the above example */

//your declared local property references
var myLeaderboard:Array;
var myUser:Object;

//your game init procedure
function initGame():void {
  //your other init codes here ...
  mytoolkit.AGgetUser(myUserReturn);
}

//your declared user data return handler
function myUserReturn(userObject:Object):void {
  myUser = userObject;
  if(myUser != null) {
    myToolkit.AGgetLeaderboard(myLeaderboardReturn, 'Friends', 'AllTime');
  }else{
    startMyGame(); //in this case the user has either not played before, or is not logged in
  }
}

//your declared leaderboard data return handler
function myLeaderboardReturn(userArray:Array):void {
  myLeaderboard = userArray;
  startMyGame(); //example method to invoke after the return is complete
}
```

There are several ways to present and use the Leaderboard data, for instance tracking the users current position in the Leaderboard during gameplay.

```
//your declared score statistic
var score:Number = 0;

//example leaderboard comparison method - returns runtime rank
function getUserPosition():int {
  if(myLeaderboard != null) {
    for(var i=0; i<myLeaderboard.length; i++) {
      if(score >= myLeaderboard[i].score) return (i+1);
    }
  }
  return (myleaderboard.length+1); //currently last position
}

//example leaderboard comparison method - returns runtime next rank username
function getNextUsername():String {
  if(myLeaderboard != null) {
    for(var i=0; i<myLeaderboard.length; i++) {
      if(score >= myLeaderboard[i].score) {
       return (i-1 < 0) ? myUser.name : myLeaderboard[i-1].name;
      }
    }
  }
  return myLeaderboard[myLeaderboard.length-1].name;
}
```

AGtoolkit: