Appendix A. Quick References

Table of Contents

A.1 TorqueScript Quick Reference

A.1.1. Conventions

Throughout this document, for succinctness, I will refer to the Torque Game Engine Scripting language simply as Torque Script.

A.1.2. Syntax/Rules

Torque Script is a typeless script that is very similar in syntax to C/C++. You will find that most C/C++ operators function as expected in Torque Script. In addition to providing the strengths of C/C++, Torque Script provides:

- Auto creation and destruction of local/global variables and their storage.
- String catenation, comparison, and auto-string-constant creation (see 'Literals' below).
- Function packaging.

A.1.3. Literals

Numbers	123	integer
	1.23	floating-point
	1.00E-003	scientific notation
	0xabc	hexadecimal
Strings	"abcd"	string
	'abcd'	tagged string
String Operators	@	catenation
	TAB	tab catenation
	SPC	space catenation
	NL	newline catenation
Escape Sequences	\n	newline
	\r	carriage return
	\t	tab
	\c0 .. \c9	colorize subsequent console output
	\cr	reset to default color
	\cp	push color from color stack
	\co	pop color from color stack
	\xhh	two-digit hex value ASCII code
	\\\\	backslash
booleans	TRUE	1
	FALSE	0

arrays	$MyArray[n]	single-dimension
	$MyMultiArray[n,m]	multi-dimension
	$MyMultiArray**n_m**	multi-dimension
vectors	"1.0 2.0 1.0 2.0"	4-element vector

A.1.4. Operators in TGE

Operator	Name	Example	Explanation
Variable Operators			
$	global	$a	$a is a global variable.
%	local	%b	%b is a local variable.
Arithmetic Operators			
*	multiplication	$a * $b	Multiply $a and $b.
/	division	$a / $b	Divide $a by $b.
%	modulo	$a % $b	Remainder of $a divided by $b.
+	addition	$a + $b	Add $a and $b.
-	subtraction	$a - $b	Subtract $b from $a.
++	auto-increment (post-fix only)	$a++	Increment $a after use. Note: ++$a is illegal.
- -	auto-decrement (post-fix only)	$b--	Decrement $b after use. Note: --$b is illegal.
Relations and Logical Operators			
<	Less than	$a < $b	1 if $a is less than % b 0 otherwise.
>	More than	$a > $b	1 if $a is greater than % b 0 otherwise.
<=	Less than or Equal to	$a <= $b	1 if $a is less than or equal to % b 0 otherwise.
>=	More than or Equal to	$a >= $b	1 if $a is greater than or equal to % b 0 otherwise.
==	Equal to	$a == $b	1 if $a is equal to % b 0 otherwise.
!=	Not equal to	$a != $b	1 if $a is not equal to % b 0 otherwise.
!	Logical NOT	!$a	1 if $a is 0 0 otherwise.
&&	Logical AND	$a && $b	1 if $a and $b are both non-zero 0 otherwise.
||	Logical OR	$a || $b	1 if either $a or $b is non-zero 0 otherwise.
Bitwise Operators			
~	Bitwise complement	~$a	flip bits 1 to 0 and 0 to 1. (i.e. ~10b == 01b)
&	Bitwise AND	$a & $b	composite of elements where bits in same position are 1. (i.e. 1b & 1b == 1b)

Operator	Name	Example	Explanation
|	Bitwise OR	$a | $b	composite of elements where bits 1 in either of the two elements. (i.e. 100b & 001b == 101b)
^	Bitwise XOR	$a ^ $b	composite of elements where bits in same position are opposite. (i.e. 100b & 101b == 001b)
<<	Left Shift	$a << 3	element shifted left by 3 and padded with zeros. (i.e. 11b << 3d == 11000b)
>>	Right Shift	$a >> 3	element shifted right by 3 and padded with zeros. (i.e. 11010b >> 3d == 00011b)

Operator	Name	Example	Explanation
	Assignment Operators		
=	Assignment	$a = $b;	Assign value of $b to $a.
op=	Compound Assignment	$a ***op***= $b;	Equivalent to $a = $a ***op*** $b. ***op*** can be any of: * / % + - & | ^ << >>
	String Operators / Constants		
@	String catenation	$c @ $d	Concatenates strings $c and $d into a single string. Numeric literals/variables convert to strings.
NL	New Line	$c NL $d	Same as catenation example with new-line between $c and $d.
TAB	Tab	$c TAB $d	Same as catenation example with tab between $c and $d.
SPC	Space	$c SPC $d	Same as catenation example with space between $c and $d.
$=	String equal to	$c $= $d	1 if $c equal to $d .
!$=	String not equal to	$c !$= $d	1 if $c not equal to $d.
	Miscellaneous Operators		
? :	Conditional	x ? y : z	Substitute y if x equal to 1, else substitute z.
[]	Array element	$a[5]	Sixth element or array $a
.	Field/Method selection	%obj.field %obj.method()	Select a console method or field
( )	Grouping	--	--
{ }	Blocking	--	--
,	Listing	--	--
::	Namespace	Item::onCollision()	This definition of the onCollision() function is in the Item namespace.
" "	String constant (normal)	"Hello world"	This is a normal string.
			This is a tagged string. The value of a tagged string is sent only once to a client

Operator	Name	Example	Explanation
` `	String constant (tagged)	‘Torque Rocks’	from the server (across the network). Subsequently, only the ‘tag’ is sent. All clients will store the string at and index identified by the ‘tag’ and can look up the value instead of requiring it be sent repeatedly.
//	Single line comment	// This is a comment	Used to comment out a single line of TS.
/* */	Multi-line comment	/* This is a a multi-line comment */	Used to comment out multiple consecutive lines. **/*** opens the comment , and ***/** closes it.

A.1.5. Keywords

break	**case**	**continue**	**datablock**
default	**else**	**FALSE**	**for**
function	**if**	**new**	**or**
parent	**return**	**switch**	**switch$**
TRUE	**while**		

Note: Although keywords are not reserved, it is considered bad practice to use variables that have the same spelling as a keyword.

break

Purpose
Use *break* to exit the innermost *for* or *while* loop. *break* can also be used to exit a switch or switch$ statement.

```
%count = 0;
while( 1 )
{
    echo(%count++);
    if (%count > 2)
    break;
}
```

See Also
case, if, switch, switch$, while

case

Purpose
Used to label cases in a *switch* or *switch$* statement.

See *switch* and *switch$* examples.

See Also
break, switch, switch$

continue

Purpose
The *continue* keyword causes the script to skip the remainder of the innermost loop in which it appears;

```
%count = 0;
while(%count++ < 8)
{
    if (%count > 2) continue;
    echo(%count);
}
```

See Also:
for, while

datablock

Purpose
The datablock keyword is used to declare a new datablock. A datablock object is used in the declaration and initialization of a special set of classes that take a datablock(s). Datablocks are created on the server and ghosted to clients.

Syntax

```
datablock DatablockClass ( NewDatablockName : InheritDatablock )
{
    className = "SomeName";

    DataBlockBody
};
```

- *DatablockClass* – A predefined engine class which inherits from SimDataBlock or one of it's children.
- *NewDatablockName* – The name for this datablock. Must be unique.
- *InheritDatablock* – Previously defined datablock to inherit (copy) *DataBlockBody* values from. Optional.

- *className* – This is a special field which can be initialized with the name of a previously defined [in Torque Script] datablock. In effect, this inserts the '*SomeName*' name into the namespace calling sequence between *NewDatablockName* and *DatablockClass*. For some datablock classes, *className* can be a non-datablock name, but it isn't guaranteed to work for all calling sequences or classes.

- *DataBlockBody* – Fields and the values they will be assigned for this datablock.

```
datablock SimDataBlock( myDataBlock )
{
    newField = "Hello";
    newField2 = 10;
};
```

Here we have declared a new SimDataBlock datablock named myDataBlock. Additionally, we have given it a new field named newField, initialized to "Hello" and a new field named newField2 initialized to 10. The namespace calling sequence for this datablock is:
myDataBlock -> SimDataBlock

```
datablock SimDataBlock( mySecondDataBlock : myDataBlock)
{
    className = "myDataBlock";
    newField2 = 15;
};
```

Here we have declared a new SimDataBlock datablock named mySecondDataBlock that derives from myDataBlock. Because it is deriving from a prior datablock, it will copy any fields that were declared and/or initialized in the 'parent' datablock. However, because we are re-declaring the field newField2. The new initialization value is taken in preference to the copied value, meaning that newField has the value "Hello" and newField2 has the value 15. Finally, we have defined className as myDataBlock, making the namespace calling sequence for mySecondDataBlock:

mySecondDataBlock -> myDataBlock -> SimDataBlock

See Also
new, Parent

Current Datablock Classes and associated ObjectClasses

DataBlock Class	Object Class
CameraData	Camera
DebrisData	Debris
ExplosionData	Explosion
FlyingVehicleData	FlyingVehicle
GameBaseData	GameBase
HoverVehicleData	HoverVehicle
ItemData	Item
LightningData	Lightning
MissionMarkerData	MissionMarker
ParticleData	Particle
ParticleEmitterData	ParticleEmitter
ParticleEmitterNodeData	ParticleEmitterNode
PathCameraData	PathCamera
PlayerData	Player
PrecipitationData	Precipitation
ProjectileData	Projectile
ShapeBaseData	ShapeBase
SimDataBlock	- none -
SplashData	Splash
StaticShapeData	StaticShape
TriggerData	Trigger
VehicleData	Vehicle
WheeledVehicleData	WheeledVehicle
WheeledVehicleSpring	WheeledVehicle
WheeledVehicleTire	WheeledVehicle

default

Purpose
This labels the default case in a *switch* or *switch$* statement. i.e. the case that is executed if no other cases matches the *switch*/*switch$* value.

Note: The *default* keyword must be listed after all *case* keywords. It is a syntax error to place it before subsequent *case* keywords.

See *switch* and *switch$* examples.

See Also
break, switch, switch$

else

Purpose
The *else* keyword is used with the *if* keyword to control the flow of a script. The general form of the well known if-then-else construct is as follows:

```
if (expression)
{
    statement(s);
}
else
{
    alternate statement(s);
}
```

Where the alternate statement(s) are executed if the expression evaluates to 0.

See *if* example.

See Also
if

false

Purpose
The *false* keyword is used for boolean comparison and evaluates to 0.

```
if( false == 0 )
{
    echo( "false evaulates to" SPC 0 );
}
```

See Also
if, true

for

Purpose
The *for* keyword is a looping construct whose general form is:

```
for ( expression0 ; expression1 ; expression2 )
{
    statements(s);
}
```

- expression0 – usually of the form: variable = value
- expression1 **–** usually of the form: variable *compare op* value
- expression2 – usually of the form: variable *op* OR variable *op* value

The loop continues to execute statement(s) until expression0 evaluates to zero.

Note: expression0, expression1, and expression2 are all required. If you absolutely need expression0 or expression2 to be empty just insert a 0.

Note2: Composite expressions of the form (sub_expression0 , sub_expression1 , … sub_expressionN) are illegal.

```
// Example 1
for( %val = 0 ; %val <  3 ; %val++ )
{
    echo( %val );
}


// Example 2 – 'Empty' expression 0 and 2
%value = 0;
for( 0 ; %val < 3 ; 0 )
{
    echo( %val );
    %val ++;
}

// Example 3 – Illegal sub-expressions
// This would produce an error while 'compiling'
%val = 0;
for( %val0 = 0 , %val1 = 3 ; %val0 <  3 ; %val0++, %val1-- )
{
    echo( %val0 );
    echo( %val1 );
}
```

See Also
break, continue, while

function

Purpose
The *function* keyword is used to define a new console function or method. Unlike procedural language, functions are not declared in one place and defined in another, but defined only. Redeclaring a named function later in script over-rides the previous definition. the format of a function takes one of two basic forms:

```
// function definition
function func_name( [arg0] , … , [argn] )
{
    statements;

    [return val;]
}
```

- func_name – Name by which this function will be subsequently called.
- [arg0] , … , [argn] – Optional arguments.
- statements – Any number of statements may be contained within the body of the function.
- val – Functions may optionally return a value;

```
// console method definition
function namespace::func_name( %this, [arg0] , … , [argn] )
{
    statements;
    [return val;]
}
```

- namespace – The name of a datablock or object classname.
- :: – Namespace resolution operator.
- %this – The first argument of a console method is always the handle to the object which is calling the method.

```
// A simple function
function test( %val )
{
   echo( "test(" SPC %val SPC ")" );

   if( 10 = %val ) return true;

   return false;
}

// A simple console method
function Item::test( %this , %val )
{
   echo( "Item::test(" SPC %this SPC "," SPC %val SPC ")" );

   if( 10 = %val ) return true;

   return false;
}

...
```

```
%obj = new Item()
{
   // ..
};
%obj.test(10);                  // Normal method of calling
Item::test(%obj, 10);           // Alternate method of calling
```

See Also
none.

if

Purpose
The *if* keyword is used with or without the *else* keyword to control the flow of a script. The general form of the well known if-then-else construct is as follows,

```
if (expression) {
    statement(s);
} else {
    alternate statement(s);
}
```

Where the statement(s) are executed if the expression evaluates to a non-zero value.

```
if(0)
{
    echo( “hello” );
}
else
{
    echo( “goodbye” );
}

if(5)
{
    echo( “hello” );
}
else
{
    echo( “goodbye” );
}
```

See Also
else

new

Purpose
The *new* keyword is used to instantiate (create) a new copy of a conobject. A conobject is:

- an engine class available in the console (Item(), Player(), etc.), or
- a datablock (derived or otherwise).

```
// New non-datablock (using) object
%obj = new ScriptObject();

//New datablock (using) object
datablock ItemData( GoldCoin )
{
...
};

%coin = new Item( myGoldCoin )
{
   // ...
   datablock = GoldCoin;
};
```

See Also
datablock

package

Purpose
The package keyword tells the console that the subsequent block of code is to be declared but not loaded. Packages provide dynamic function-polymorphism in TorqueScript. In short, a function defined in a package will over-ride the prior definition of a same named function when the is activated. When the package is subsequently deactivated, the previous definition of any overridden functions will be re-asserted.

Packages have the following syntax:

```
package package_name
{
          function function_definition0()
          {
                    [statements;]
          }
          ...
          function function_definitionN()
          {
                    [statements;]
          }
};
```

Things to know:

- The same function can be defined in multiple packages.
- Only functions can be packaged.
- Datablocks cannot be packaged.
- Packages 'stack' meaning that deactivating packages activated prior to the currently active (s) will deactivate all packages activated prior to the being deactivated (see example below).
- Functions in a may activate and deactivate packages.

Activating
In order to use the functions in a package, the package must be activated:

```
ActivatePackage(_name);
```

Deactivating
Subsequently a package can be deactivated:

```
DeactivatePackage(_name);
```

```
function testFunction()
{
    echo( "testFunction() - unpackaged." );
}

package MyPackage0
{
   function testFunction()
   {
       echo( "testFunction() - MyPackage0." );
   }
};
```

```
 MyPackage1
{
   function testFunction()
   {
       echo( “testFunction() – MyPackage1.” );
   }
};

...

testFunction();
// prints => testFunction() - unpackaged.

ActivatePackage( MyPackage0 );

testFunction();
// prints => testFunction() - MyPackage0.

ActivatePackage( MyPackage1 );

testFunction();
// prints => testFunction() – MyPackage1.

DeactivatePackage( MyPackage0 );
// MyPackage1 is automatically deactivated.

testFunction();
// prints => testFunction() - unpackaged.
```

See Also
function, Parent

Parent

Purpose
The *Parent* keyword is used with the namespace operator (::) to reference the previous definition of a function what has been over-ridden either through inheritance or packaging. The *Parent* keyword can only be used within specific contexts:

- From within a consoleMethod, or
- from within a packaged function.

```
// Calling an inherited parent
datablock ItemData( GoldCoin )
{
...
};

function ItemData::onAdd( %db, %obj )
{
    echo( “ItemData::onAdd()” );
}

function GoldCoin::onAdd( %db, %obj )
{
    Parent::onAdd( %db, %obj );

    echo( “GoldCoin::onAdd()” );
}

// Calling a  parent
function testFunction()
{
    echo( “testFunction() - unpackaged.” );
}

 MyPackage0
{
   function testFunction()
   {
      Parent::testFunction();
      echo( “testFunction() - MyPackage0.” );
   }
};

...

testFunction();
// prints => testFunction() - unpackaged.

ActivatePackage( MyPackage0 );

testFunction();
// prints => testFunction() - unpackaged.
// prints => testFunction() - MyPackage0.
```

See Also
datablock, function

return

Purpose
The *return* keyword is used to return a value from a *function*

```
function equal_to( %arg0 , %arg1 )
{
    return ( %arg0 == %arg1 );
}

echo( equal_to(10,11) ); // prints 0

echo( equal_to(11,11) ); // prints 1
```

See Also
function

switch

Purpose
The *switch* keyword is used to control the flow of a script. The general form of a switch statement is as follows:

```
switch (expression) {
case value0:
    statement(s);
    break;
case value1:
    statement(s);
    break;
. . .
case valueN:
    statement(s);
    break;
default:
    statement(s);
}
```

Where expression is evaluated and the subsequently compared to the following case values. If a case matches the evaluated expression, the statement(s) associated with that case are executed. If no values match and a default statement exists, the statement(s) in the default case will be executed.

switch is used ONLY for expressions that evaluate to a numeric value.

Note: Unlike C/C++, the break statements in switches are superfluous. Torque Script will only execute matching cases and NOT automatically execute all subsequent cases. This is shown in the example below.

```
%tmp = 1;

switch( %tmp  )
{
case 0:
    echo( 0 );
case 1:
    echo( 1 );
default:
    echo( "proof" );
}
```

See Also
break, case, default, switch$

switch$

Purpose
The *switch$* keyword is used to control the flow of a script. The general form of a switch statement is as follows:

```
switch (expression)
{
case string_value0:
    statement(s);
    break;
case string_value1:
    statement(s);
    break;
. . .
case string_valueN:
    statement(s);
    break;
default:
    statement(s);
}
```

Where expression is evaluated and subsequently compared to the following case values. If a case string_value matches the evaluated expression, the statement(s) associated with that case are executed. If no values match and a default statement exists, the statement(s) in the default case will be executed.

switch$ is used ONLY for expressions that evaluate to a string value.

Note: Unlike C/C++, the break statements in switches are superfluous. Torque Script will only execute matching cases and NOT automatically execute all subsequent cases.

```
%tmp = “hi” ;

switch$( %tmp )
{
case “bye”:
    echo( “bye” );
case “hi”:
    echo( “hi” );
}
```

See Also
break, case, default, switch

true

Purpose
The *true* keyword is used for boolean comparison and evalulates to 1.

```
if(true == 1)
{
    echo( “true evaulates to” SPC 1 );
}
```

See Also
if, true

while

Purpose
The *while* keyword is a looping construct whose general form is:

```
while (expression) {
    statements(s);
}
```

Where expression is usually of the form: variable *compare op* value, and the loop continues to execute statement(s) until expression evaluates to false (i.e. 0).

```
%val=5;

while( %val )
{
    echo( %val-- );
}
```

See Also
break, continue, for

A.1.5. Engine Interfacing

The engine provides a concise set of tools to expose the core engine functionality and structures in the console (to Torque Script). The following problem-solution matrix summarizes the things we may want to do and how to do them.

Problem (Want to…)	**Solution**

Expose **Member**
 	
as **Field**.
	addField()
	addFieldV()

Expose **Member**
 	
as **Field**.
	addNamedField()
	addNamedFieldV()

Expose/Remove
global **C++ Variable** or
static **Member**
 	
as **Local Variable**
	Con::addVariable()
	Con::removeVariable()

Create **Console Method** from C++.	ConsoleMethod()
Create **Console Function** from C++.	ConsoleFunction()

addField()	**addFieldV()**	**addNamedField()**	**addNamedFieldV()**
Con::addVariable()	**Con::removeVariable()**	**ConsoleFunction()**	**ConsoleMethod()**

addField

Purpose
The *addField* function provides a means to expose C++ class members as Torque Script object fields. This function is normally called in a class' initPersistFields() method.

Syntax

```
addField( const char *       in_pFieldname,
          const U32          in_fieldType,
          const dsize_t      in_fieldOffset,
          const char *       in_pFieldDocs )

addField( const char*        in_pFieldname,
          const U32          in_fieldType,
          const dsize_t      in_fieldOffset,
          const U32          in_elementCount,
          EnumTable *        in_table,
          const char *       in_pFieldDocs )
```

- **in_pFieldname** - String specifying variable name as used in console.
- **in_fieldType** - The variable type. (Types specified in consoleTypes.h).
- **in_fieldOffset** - This is a numeric value calculated using the Offset() macro.
- **in_elementCount** - Number of elements at offset. The default value is 1, but if you are referencing an array then this value will be the number of elements in the array.
- **in_table** - This argument is used when the field type is *TypeEnum*. In this special case, you need to define an *EnumTable* containing a map of the ENUM values and the strings to represent them in the console.
- **in_pFieldDocs** - A short string describing the field in plain English. This field is used in Torque's automatic Console documentation functionality.

```
class GuiCrossHairHud : public GuiBitmapCtrl
{
          ...
   ColorF   mDamageFillColor; // C++ declaration
          ...
}

void GuiCrossHairHud::initPersistFields()
{
          ...
   // Added here
   addField( "damageFillColor",
             TypeColorF,
             Offset( mDamageFillColor , GuiCrossHairHud ));
          ...
}
```

See Also
addFieldV, addNamedField, addNamedFieldV

addFieldV

Purpose
The *addFieldV* function serves the same purpose as *addField*, but introduces the concept of a 'Type Validator' classes.

Type Validator Classes are used to restrict the values a field may be given. If illegal values are assigned to a validated field, the validator function will print an error to the console and set the field to a valid value.

The two most commonly used Type Validator Classes are *FRangeValidator* and *IRangeValidator*. New validators may be derived from either of these, or their base class *TypeValidator*.

Note: *addFieldV* does not support arrays or enumerated types.

Syntax
```
addFieldV( const char         * in_pFieldname,
           const U32            in_fieldType,
           const dsize_t        in_fieldOffset,
           TypeValidator      * v )
```

- *** in_pFieldname** - String specifying variable name as used in console.
- **in_fieldType** - The variable type. (Types are specified in consoleTypes.h).
- **in_fieldOffset** - This is a numeric value calculated using the Offset() macro.
- *** v** - This is a pointer to a TypeValidator class instance of which there are several derived types to choose from.

```
// from projectile.cc around line 126
// light radius is limited to between
// 1.0 and 20.0, inclusive.

addFieldV( "lightRadius",
           TypeF32,
           ProjectileData,
           new FRangeValidator(1, 20) );
```

See Also
addField, addNamedField, addNamedFieldV

addNamedField

Purpose
The *addNamedField* is a macro that makes use of the simple version *addField* function and provides a short-hand method of adding fields. It does not support enumerated types or arrays. Also, it does not provide a means of adding documentation strings. It is worth noting that the macro uses the same name for the field as the variable.

Syntax

addNamedField(fieldname , type , className) ;

- **fieldname** - Name of variable to expose. Will be used as name of field.
- **type** - The variable type. (Types are specified in consoleTypes.h).
- **className** - Name of C++ class containing variable.

```
addNamedField( isBallistic , TypeBool , ProjectileData );
```

See Also
addField, addFieldV, addNamedFieldV

addNamedFieldV

Purpose
The *addNamedField* is a macro that makes use of the *addFieldV* function and provides a short-hand method of adding validated fields.

Syntax
addNamedFieldV(fieldname , type, className, validator);

- **fieldname** - Name of variable to expose. Will be used as name of field.
- **type** - The variable type. (Types are specified in consoleTypes.h).
- **className** - Name of C++ class containing variable.
- **validator** - This is a pointer to a TypeValidator class instance of which there are several derived types to choose from.

```
addNamedFieldV( lightRadius,
                TypeF32,
                ProjectileData,
                new FRangeValidator( 1 , 20 ) );
```

See Also
addField, addFieldV, addNamedField

Con::addVariable

Purpose
The *Con::addVariable* function provides a means to expose global C++ variables or static class Members as Torque Script global variables. This function is normally called in a class' consoleInit () method.

Syntax
Con::addVariable(const char * name , t , void * dp);

- **name** - String specifying name of global variable (in console).
- **t** - The variable type. (Types are specified in consoleTypes.h).
- **dp** - A pointer to the global C++ variable, or static Member.

```
// From camera.cc
Con::addVariable( "Camera::movementSpeed" , TypeF32 , &mMovementSpeed );
```

See Also
Con::removeVariable

Con::removeVariable

Purpose
The *Con::removeVariable* function provides a means to un-expose global C++ variables or static class Members previously exposed as Torque Script global variables with *Con::addVariable*.

This call is global. i.e. Once we remove a variable, we cannot put it back.

Syntax
removeField(const char* in_pFieldname);

- **in_pFieldname** - String specifying field to be removed.

```
// 1. TerrainBlock is inherited from SceneObject.
// 2. SceneObject links member mObjToWorld to the
//    Torque Script field position.
// 3. TerrainBlock undoes this(in terrData.cc)

removeField( "position" );
```

See Also
Con::addVariable

ConsoleFunction

Purpose
The *ConsoleFunction* macro provides a means to create function from C++ in the console.

Syntax
```
ConsoleFunction(  name , returnType , minArgs , maxArgs , usage )
{

// Function body

}
```

- **name** - This is the name of the function as it will be used in the console.
- **returnType** - Is the return type of the function.
- **minArgs** - Minimum arguments this function can accept. 1 is the minimum, because the name of the function is automatically passed as the first argument.
- **maxArgs** - Maximum arguments this function can accept. If you put 0 in this field, it means any number of arguments may be passed to the function.
- **usage** - Is a string that will be printed as a help statement if someone later attempts to use this function with the wrong number of arguments.

```
// From main.cc
ConsoleFunction( getSimTime , S32 , 1 , 1,
                 "getSimTime() - Time since game started.")
{
   return Sim::getCurrentTime();
}
```

See Also
ConsoleMethod

ConsoleMethod

Purpose
The *ConsoleMethod* macro provides a means to create Console Method from C++ in the console. The static variant of ConsoleMethod is for methods that you want to be able to call call statically. For example:, *GameConnection::getServerConnection()*.

Syntax
```
ConsoleMethod( className , scriptname , returnType ,
               minArgs , maxArgs , usage )
{

    // Method body
}

// or

ConsoleStaticMethod( className , scriptname , returnType ,
                     minArgs , maxArgs , usage )
{
          // Method body
}
```

- **className** - The name of the class the method is in.
- **scriptname** - The name the method will be given in the console (i.e. used by TorqueScript).
- **returntype** - The return type of the method.
- **minargs** - The minimum arguments this method takes. The minimum is 2, because the name of the console method is automatically passed as the first argument and the object's handle is automatically passed as the second argument.
- **maxargs** - The maximum number of args that can be passed to this method. If you put 0 in this field, it means any number of arguments may be passed to the method.
- **usage** - A string that will be printed as a help statement if someone later attempts to use this method with the wrong number of arguments. This usage is also when you use the obj.*dump()* command.

```
//From SimBase.cc
ConsoleMethod( SimObject , getId , S32 , 2 , 2 ,
               "obj.getId()" )
{
   argc; argv;
   return object->getId();
}
```

See Also
ConsoleFunction

A.2 Console Objects Fields and Methods Quick Reference

A.2.1. ActionMap

Console Method Summaries

bind	**bindCmd**	**getBinding**
getCommand	**getDeadZone**	**getScale**
isInverted	**pop**	**push**

Console Methods

bind(device , action , [modifier , mod...] , command)

Purpose
Use the ***bind*** method to associate a function to a keystroke or other device input.

Syntax
```
  device - Name of the device to bind the command to (see 'Device Table' below).
  action - Name of the action to watch for(see 'Action Table' below).
modifier - Special modifiers (mouse only), such as dead spot, etc.
command  - The function to be called on make and break.
```

Returns
No return value.

Notes
The ***command*** bound via the bind function must be specified as a flat name with no elipses or semi-colon termination and will be called on make and break events (i.e. key press and release for a mapped key).

Args:

Warning: When a function is bound to a keystroke or other device input, and no other versions of the binding are provided, the function will be called even if a modifier key like CTRL, ALT, or SHIFT is also pressed. For clarification, see 'Bind Sample' example below.

See Also
bindCmd, getBinding, unbind

bindCmd(device , action , makeCmd , breakCmd)

Purpose
Use the ***bindCmd*** method to associate up to two functions to a keystroke or other device input.

Syntax
```
  device - Name of the device to bind the command to (see 'Device Table' below).
  action - Name of the action to watch for(see 'Action Table' below).
 makeCmd - The function to be called on make event.
breakCmd - The function to be called on break event.
```

Returns
No return value.

Notes
The ***makeCmd*** is bound to the make event and the ***breakCmd*** is bound to the break event and in both cases, the commands are specified as complete scripts, with all arguments, elipses, and the terminating semi-colon. Either of these commands may be non-specified (NULL strings). For clarification, see 'Bind Sample' example below.

See Also
bind, getBinding, unbind

getBinding(command)

Purpose
Use the ***getBinding*** method to get the binding for a specified ***command***.

Syntax
command – The function to seek a binding for.

Returns
Returns a string containing the binding as a field (TAB separated string), or a NULL string meaning 'no binding found'.

See Also
bind, bindCmd

getCommand(device , action)

Purpose
Use the ***getCommand*** method to get the function associated with a specific ***device*** + ***action*** pair.

Syntax
device – Name of the device to bound to a command (see 'Device Table' below).
action – Name of the action to watch for (see 'Action Table' below).

Returns
Returns the function name or specification associated with the specified ***device*** + ***action*** pair, or a NULL-string meaning 'no binding found'.

See Also
bind, bindCmd, getBinding

getDeadZone(device , action)

Purpose
Use the ***getDeadZone*** method to get the dead-zone associated with a specific ***device*** + ***action*** pair.

Syntax
device - Name of the device to bound to a command (see 'Device Table' below).
action - Name of the action to watch for (see 'Action Table' below).

Returns
Returns a dead-zone specification, or "0 0" meaning that there is no dead-zone, or a NULL string meaning the mapping was not found.

See Also
bind, bindCmd

getScale(device , action)

Purpose
Use the ***getScale*** method to get the scale associated with a specific ***device*** + ***action*** pair.

Syntax
device - Name of the device to bound to a command (see 'Device Table' below).
action - Name of the action to watch for (see 'Action Table' below).

Returns
Returns 1 if no scale is associated with the specified ***device*** + ***action*** pair, or the mapping was not found.

See Also
bind, bindCmd

isInverted(device , action)

Purpose
Use the ***Purpose*** method to determine if a specific ***device*** + ***action*** pair in inverted.

Syntax
device - Name of the device to bound to a command (see 'Device Table' below).
action - Name of the action to watch for (see 'Action Table' below).

Returns
Returns 1 if the mouse (or other scrolling device) is inverted, 0 otherwise.

Notes
This only applies to scrolling devices.

See Also
bind, bindCmd

pop()

Purpose
Use the ***pop*** method to de-activate an ActionMap and remove it from non-global ActionMap stack.

Returns
No return value.

See Also
push

push()

Purpose
Use the ***pop*** method to activate an ActionMap and place it at the top of the non-global ActionMap stack.

Returns
No return value.

See Also
pop

save([fileName] [, append])

Purpose
Use the ***save*** method to save an entire action map specification to a file. If ***append*** is not specified, or specified as false, ***fileName*** will be overwritten, otherwise the action map will be appended to the file.

Syntax
fileName - Full path to file in which to store action map definition.
append - If true, do not overwrite the file, else start from scratch.

Returns
No return value.

unbind(device , action)

Purpose
Use the ***unbind*** method to remove a previosly specified ***device*** + ***action*** pair from the action map.

Syntax
device - Name of the device to bound to a command (see 'Device Table' below).
action - Name of the action to watch for (see 'Action Table' below).

Returns
No return value.

See Also
bind, bindCmd

Device Table

Device	Description
keyboard**N**	This is the **N**th keyboard hooked up to the system. For the first keyboard, either "keyboard" or "keyboard0" is acceptable.
mouse**N**	This is the **N**th mouse hooked up to the system. For the first mouse, either "mouse" or "mouse 0" is acceptable.
joystick**N**	This is the **N**th joystick **or gamepad** hooked up to the system.
unkown**N**	This is the **N**th unknown device up to the system. In other words, some devices has been sampled, but TGE doesn't know what it is.

Action Table

Action	Description
button0, buton1, ... , button31	This is a mouse, joystick, or gamepad button press. For the mouse, buttons 0,1, and 2 are left, right, and middle buttons respectively. See the appendix for other button mappings.
a .. z A .. Z 0..9 F1..F12	These are keyboard inputs. Because this list is so long and in order to accommodate possible variances for special keyboards and other devices a same GUI has been provided with the kit that displays the current action, be it keyboard, mouse, joystick/gamepad, or other device. Simple start the kit and click SampleGUIs -> Input. Follow the instructions provided in the sample.
shift ctrl alt	These are modifiers and are not used standalone, but they are included in the action string, for example: "shift p" is the shift key and the p key pressed at the same time.
lshift, rshift, lctrl, rctrl, lalt, ralt	These are special modifier actions. Theye only register as 'break' events when one of these keys: left shift, right shift, left ctrl, right ctrl, left alth, or right alt is released.

Mouse Modifiers

Action Modifiers	Description
D %x %y	Has dead zone. This is used to add a dead zone for the mouse. Motions in this zone will not be recorded. This can be used to remove the jitter caused by a ‘nervous hand’.
S %s	Has Scale. This is used to scale the mouse motion (by a multiple).
I	Inverted. This is used to invert the mouse.
R %s	Has Scale. Same as S.

A.2.2. AIConnection

This is a derivative of the class GameConnection (described later) and thus inherits all of that objects methods. Additionally, it adds features that allow this connection type to be driven by AI scripts.

Console Method Summaries

getAddress	**getFreeLook**	**getMove**
getTrigger	**setFreeLook**	**setMove**

Console Methods

getAddress()

Purpose
Use the ***getAddress*** method to get the address an AIConnection is currently connected to.

Returns
Returns the address of the current connection in the format: " A.B.C.D:Port ", where A .. B are standard IP numbers between 0 and 255 and Port can be between 1000 and 65536.

getFreeLook()

Purpose
Use the ***getFreeLook*** method to check if the current connection is in free look mode.

Returns
Returns true if the current connection is free look mode.

See Also
setFreeLook

getMove(field)

Purpose
Use the ***getMove*** method to get the move setting for **field**.

Syntax
field - Move setting to check.

Returns
Returns a value between 0.0 and 1.0.

See Also
getTrigger, setMove, setTrigger

getTrigger(trigger)

Purpose
Use the ***getTrigger*** method to get the current value for a trigger specified by the numeric value trigger.

Syntax
trigger – An integer value between 0 and 6 representing on of the seven triggers.

Returns
Returns 1 if the trigger is active/depressed, and 0 if it is not.

See Also
getMove, setMove, setTrigger

setFreeLook(isFreeLook)

Purpose
Use the ***setFreeLook*** method to set the current connection to a free look mode.

Syntax
isFreeLook – A boolean value. If true, freelook will be enable, otherwise it will be disabled.

Returns
No return value.

See Also
getFreeLook

setMove(field , value)

Purpose
Use the ***setMove*** method to set a move type for the control-object connected to this AIConnection.

Syntax
field – A string containing a move type. See 'move types' table below.
value – A value between 0.0 and 1.0.

Returns
No return value.

See Also
getMove, getTrigger, setTrigger

setTrigger(trigger , set)

Purpose
Use the ***setTrigger*** method to set or un-set a trigger on the control-object connected to this AIConnection.

Syntax
```
trigger – An integer value between 0 and 6 representing on of the seven triggers.
    set – A boolean value.  If true, this trigger is enabled/set/depressed,
          otherwise, it is disabled/un-set/released.
```

Returns
No return value.

See Also
getMove, getTrigger, setMove

A.2.3. AIPlayer

This class is used to represent a basic AI driven player model. It has all the fields and methods of the player class. It adds several new methods to allow basic navigation and aiming. Most AI behaviors come from scripts using this small set of methods.

Console Method Summaries

clearAim	getAimLocation	getAimObject
getMoveDestination	setAimLocation	setAimObject
setMoveDestination	setMoveSpeed	stop

Console Methods

clearAim()

Purpose
Use the ***clearAim*** method to stop aiming at an object or a point.

Returns
No return value.

See Also
setAimObject, setAimLocation

getAimLocation()

Purpose
Use the ***getAimLocation*** method to get the point the bot is currently aiming at. This will reflect the position set by setAimLocation(), or the position of the object that the bot is now aiming at, or if the bot is not aiming at anything, this value will change to whatever point the bot's current line-of-sight intercepts.

Returns
Returns an XYZ vector containing the location of the bot's current aim.

See Also
setAimObject, setAimLocation

getAimObject()

Purpose
Use the ***getAimObject*** method to get the ID of the object the bot is currently aiming at.

Returns
Returns -1 if no object is being aimed at, or a non-zero positive integer ID of the object the bot is aiming at.

See Also
getAimLocation, setAimObject, setAimLocation

getMoveDestination()

Purpose
Use the ***getMoveDestination*** method to get the last set move destination.

Returns
Returns a vector containing the <x y z> position of the bot's current move destination. If no move destination has yet been set, this returns "0 0 0".

Notes
The bot will not look at its destination unless it is told to, which means it can aim at one object or location while walking in another direction.

See Also
setMoveDestination

setAimLocation(target)

Purpose
Use the ***setAimLocation*** method to set the bot's aim location to ***target***.

Syntax
target – An XYZ vector representing a position in the game world.

Returns
No return value.

See Also
getAimLocation, getAimObject, setAimObject

setAimObject(obj [, offset])

Purpose
Use the ***setAimObject*** method to set the current object for the bot to aim at, using an optional ***offset*** to modify that aim.

Syntax
```
   obj – A valid GameBase object ID or name.
offset – A three-element offset vector which will be added to the position of the
         aim object.
```

Returns
No return value.

See Also
getAimLocation, getAimObject, setAimLocation

setMoveDestination(goal [, slowDown])

Purpose
Use the ***setMoveDestination*** method to set the bot's current move destination top ***goal***. This will cause the bot to start moving immediately towards that destination.

Syntax
```
    goal – An XYZ vector containing the position for the bot to move to.
slowDown – A boolean value.  If set to true, the bot will slow down when it gets
           within 5-meters of its move destination.  If false, the bot will stop
           abruptly when it reaches the move destination. By default, this is true.
```

Returns
No return value.

Notes
Upon reaching a move destination, the bot will clear its move destination and calls to getMoveDestination will return a NULL string.

See Also
getMoveDestination, setMoveSpeed, stop

setMoveSpeed(speed)

Purpose
Use the ***setMoveSpeed*** method to modify the bot's movement rates between 0.0 (0%) and 1.0 (100%).

Syntax
speed – A speed multiplier between 0.0 and 1.0. This is multiplied by the bot's base movement rates (from its datablock).

Returns
No return value.

See Also
setMoveDestination, stop

stop()

Purpose
Use the ***stop*** method to stop the bot from moving.

Returns
No return value.

Notes
This does not clear the bot's move destination, so a call to *getMoveDestination* will be able to get it.

See Also
setMoveDestination, setMoveSpeed

A.2.4. AIWheeledVehicle

Fields

Field Name	Description	Sample or Range
disableMove	If this value is set to true, the vehicle will not respond to any movement commands.	[false , true]

Console Method Summaries

getMoveDestination	setMoveDestination	setMoveSpeed
setMoveTolerance	stop	

Console Methods

getMoveDestination()

Purpose
Use the ***getMoveDestination*** method to get the last set move destination.

Returns
Returns a vector containing the <x y z> position of the vehicle's current move destination. If no move destination has yet been set, this returns "0 0 0".

Notes
The vechicle will not look at its destination unless it is told to, which means it can aim at one object or location while walking in another direction.

See Also
setMoveDestination

setMoveDestination(goal [, slowDown])

Purpose
Use the ***setMoveDestination*** method to set the vehicle's current move destination top ***goal***. This will cause the vehicle to start moving immediately towards that destination.

Syntax
```
    goal – An XYZ vector containing the position for the vehicle to move to.
slowDown – A boolean value.  If set to true, the vehicle will slow down when it gets
           within 5-meters of its move destination.  If false, the vehicle
           will stop abruptly when it reaches the move destination. By default,
           this is true.
```

Returns
No return value.

Notes
Upon reaching a move destination, the vehicle will clear its move destination and calls to getMoveDestination will return a NULL string.

See Also
getMoveDestination, setMoveSpeed, setMoveTolerance, stop

```
Args:
   goal     - A vector containing  the <x y z> position for the vehicle to move to.
   slowDown - Slow down near destination?
```

setMoveSpeed(speed)

Purpose
Use the ***setMoveSpeed*** method to modify the vesicle's movement rates between 0.0 (0%) and 1.0 (100%).

Syntax
speed - A speed multiplier between 0.0 and 1.0. This is multiplied by the vesicle's base movement rates (from its datablock).

Returns
No return value.

See Also
setMoveDestination, setMoveTolerance, stop

setMoveTolerance(tolerance)

Purpose
Use the ***setMoveTolerance*** method to set the tolerance radius for an AIWheeled vehicle's target position. In other words, how close to the target does the vehicle have to get before it is considered 'on target'?

Syntax
tolerance - The maximum distance away from a move target the vehicle can be and still be considered 'on target'.

Returns
No return value.

Notes
Because vehicles can have major variances in size, and because of the way 'on target' is measured, this method is provided to make it easier for a wheeled vehicle to hit it's movement destination.

See Also
getMoveSpeed, setMoveSpeed

stop()

Purpose
Use the ***stop*** method to stop the vechicle from moving.

Returns
No return value.

Notes
This does not clear the vechicle's move destination, so a call to *getMoveDestination* will be able to get it.

See Also
setMoveDestination, setMoveSpeed

A.2.5. AudioDescription

Fields

Field Name	Description
coneInsideAngle	Sweep angle of inner cone.
coneOutsideAngle	Sweep angle of Outside cone.
coneOutsideVolume	Maximum gain in Zone C (see above).
coneVector	Audio emitter's eye (pointing) vector.
environmentLevel	Amount by which the audioEnvironment datablock used with this datablock will affect sound.
is3D	Is this a 2D or a 3D sound?
isLooping	If true, sound loops, otherwise sound plays only once.
isStreaming	If true, sound source is streaming, otherwise sound is from file.
loopCount	Number of times to loop this sound. • -1 – Loop infinitely. • 0 – Loop once and only once. • 1 – Loop once, possibly twice. • (N > 1) – Loop N times.
maxDistance	Outer boundary for 3D sound sphere. Sound turns on-off here at this distance from 3D emitter.
maxLoopGap	Maximum delay between subsequent loop.
minLoopGap	Minimum delay between subsequent loop.
ReferenceDistance	Distance at which sound turns on to 100% of current maximum gain.
type	Audio type. There can be multiple numeric audio types, each with its own globally controlled max gain.
volume	Maximum gain for this source.

A.2.6. AudioEmitter

Audio emitters are non-visible objects that can be placed in the game world and are used to produce 2D and 3D sound in game. All sounds produced by these emitters are networked. Therefore, all audio profiles, and other sound entities used by emitters must use the datablock keyword (not new).

Fields

Field Name	Description	Sample or Range
coneInsideAngle	Sweep angle of inner cone.	[0..360]
coneOutsideAngle	Sweep angle of Outside cone.	[0..360]
coneOutsideVolume	Maximum gain in Zone C (see above).	[0.0, 1.0]
coneVector	Audio emitter's eye (pointing) vector.	**information only (use visual feedback to direct sound)**
description	Optional Audiodescription.	AudioDescription
enableVisualFeedback	Enable visual effects showing audio-cones, facing, and on-off indication.	[false , true]
fileName	Relative or absolute path to sound file.	Filename
is3D	Enables 3D sound.	[false , true]
isLooping	Enable looping.	[false , true]
loopCount	• -1 – Loop infinitely. • 0 – Loop once and only once. • 1 – Loop once, possibly twice. • (N > 1) – Loop N times.	See Description
maxDistance	Outer boundary for 3D sound sphere. Sound turns on-off here at this distance from 3D emitter.	[referenceDistance , inf)
maxLoopGap	Maximum delay between subsequent loop.	-
minLoopGap	Minimum delay between subsequent loop.	-
outsideAmbient	If true, plays outside only (turns off when player is inside interior).	[false , true]
position	Optional audio profile.	AudioProfile
Profile	Maximum gain distance. Inner-cone gain is maxed when player is within this distance of 3D emitter.	[0.0 , maxDistance]
ReferenceDistance	Distance at which sound turns on to 100% of current maximum gain.	[0.0 , inf.0)
type	Audio type. There can be multiple numeric audio types, each with its own globally controlled max gain.	[0 , inf)
useProfileDescription	Use audio profile instead of values set via inspector.	[false , true]
volume	Emitter's maximum gain.	[0.0 , 1.0]

A.2.6. AudioProfile (AP)

This is a special type of data block used to describe 2-D and 3-D sounds. It can be created using the new keyword or the data block keyword. In the prior case, the audio description is intended to be used for 2-D sounds usually for interfaces and other client-side sounds. Space In the latter case, the audio description is intended to be used for networked sounds. This is a crucial difference.

Fields

Field Name	Description	Sample or Range
description	AudioDescription to use with this AP.	see type
environment	- ignore -	--
fileName	Path to WAV or OGG file.	~/path/filename.wav
preload	If set to true, file is pre-loaded, otherwise it is loaded on-demand.	[false , true]

A.2.8. Camera

The camera class is a lightweight class use to represent the position in view of the camera in the game world. It is derived from shape base, and adds no new fields of its own. It adds a few new methods, used to enable free fly mode and orbiting mode.

Globals

Variable Name	Description	Sample or Range
Camera::movementSpeed	Controls the free camera movement speed.	[0 , inf)

Console Method Summaries

getPosition	setFlyMode	setOrbitMode

Console Methods

getPosition()

Purpose
Use the ***getPosition*** method to find the current world position of the camera.

Returns
Return a vector containing the XYZ position of the camera.

setFlyMode()

Purpose
Use the ***setFlyMode*** method to toggle the camera between free fly mode and attached mode. In the prior mode, the camera is able to move about the game world free of an avatar, it is in effect the avatar. The latter mode, is when the camera is attached to another object, the control object.

Returns
No return value.

Notes
When if fly-mode, $Camera::movementSpeed controls the movement rate of the camera.

setOrbitMode(orbitObject , transform , minDistance , maxDistance , curDistance , ownClientObject)

Purpose
Use the ***setOrbitMode*** method to force the camera to orbit any particular game base object with a starting position and orientation and at a fixed minimum and maximum distance.

Syntax

orbitObject	– The object to orbits about.
transform	– A matrix describing both the position and orientation of the camera.
minDistance	– The minimum distance the camera is allowed to be from the object it is orbiting about.
maxDistance	– The maximum distance the camera is allowed to be from the object it is orbiting about.
curDistance	– The starting position of the camera from the object it is orbiting about. Note, this overrides the position portion of the transform, forcing the camera to be at this distance from the object.
ownClientObject	– Set this to true if the object the camera is orbiting is owned by the client that owns the camera, otherwise set to false.

Returns
No return value.

A.2.9. CameraData

This is the data block that goes with the camera. Space It is used to do find any special attributes the camera should have, if it is not in fact to deriving these attributes from the object that it is attached to.

A.2.10. ConsoleLogger

This class is supplied to allow us to capture the output of the console and redirected to a file. Is useful in a variety of situations.   it can be said to capture all output, only warnings, or only errors.

Fields

Field Name	Description	Sample or Range
level	Logging level for this logger.	normal, warning, error

A.2.11. Debris

Debris objects are used to represent the refuse left behind by an exploding or destroyed object. However, this object is versatile enough to be used for various purposes, to include a rockfall that blocks the road, the remains of a fallen building, etc.

Fields

Field Name	Description	Sample or Range
lifeTime	Lifetime of debris in seconds.	[0.0 , inf.0)

Console Methods

init()

`init( position , velocity )`

Purpose
Use the ***init*** method to set the initial ***position*** and ***velocity*** of a debris object.

Syntax
position – A three-element floating-point vector representing the starting location of the debris object in the game world.
velocity – A three-element floating-point vector representing the direction and magnitude of the debris object's original path.

Returns
Returns true on success or false on failure.

A.2.12. DebrisData

The data block that goes with debris. This class is used to describe most of the behavior of the debris.

Fields

Field Name	Description	Sample or Range
baseRadius	Debris starts at minimum of this distance from creation point. Requires useRadiusMass to be true.	(0.2 , inf.0)
bounceVariance	Total bounces == numBounces +/- bounceVariance.	(0 , numBounces)
elasticity	How bouncy is this debris? Values > 1.0 add energy to the system.	[0.0 , inf.0)
emitters[0] emitters[1]	ParticleEmitterData datablocks used to trail particles behind moving debris.	ParticleEmitterData
explodeOnMaxBounce	Does this debris explode when it hits maxBounce count?	[false , true]
Explosion	ExplosionData datablock used for exploding debris.	ExplosionData
fade	If set to true, the debris will start to fade from view in the last second of its lifetime.	[false , true]
friction	How much friction is applied when thid debris slides?	[0.0 , 1.0]
gravModifier	How much does gravity affect this debris?	(-inf.0 , inf.0)
ignoreWater	If set to false, debris will bounce off of water, else it will sink.	[false , true]
lifetime	Total life of particle == lifetime +/- lifetimeVariance in milliseconds.	[0.0 , inf.0)
lifetimeVariance	Total life of particle == lifetime +/- lifetimeVariance in milliseconds.	[0.0 , lifetime)
maxSpinSpeed	Maximum angular rotation of debris in degrees-per-second.	(-inf.0 , inf.0)
minSpinSpeed	Minimum angular rotation of debris in degrees-per-second.	(-inf.0 , inf.0)
numBounces	Total bounces == numBounces +/- bounceVariance.	[0 , inf)
render2D	If this field is set to true, debris will render a billboard using texture as the image source.	[false , true]
shapeFile	The file to be used for this debris' mesh.	~/path/filename
snapOnMaxBounce	If set to to true, and staticOnMaxBounce is also true, this debris will 'stick' in its last contact orientation on the surface it contacted.	[false , true]
staticOnMaxBounce	If set to to true, the debris will be replaced (temporarily) with a static shape. This shape will still be destroyed after the debris's total lifetime expires.	[false , true]
terminalVelocity	Maximum velocity at which this debris will fall (or rise).	[0.0 , inf.0)
texture	A texture to be used for rendering a billboard, if render2D is set to true.	~/path/filename
useRadiusMass	If set to true (and baseRadius > 0.2) , the initial rendering position of the debris will be at a random position baseRadius from the creation point.	[false , true]
velocity	Total initial velocity == velocity +/ velocityVariance.	[0.0 , inf.0)
velocityVariance	Total initial velocity == velocity +/ velocityVariance.	[0 .0 , velocity)

A.2.13. DecalData

A data block describing an individual decal.

Fields

Field Name	Description	Sample or Range
sizeX	The horizontal width in meters.	[0.0 , inf.0)
sizeY	The vertical height in meters.	[0.0 , inf.0)
textureName	The full path to the texture to use this decal.	~/path/filename

A.2.14. DecalManager

All details are managed by the decal manager which provides a few global variables for managing the number of decals that are visible at any one time and the lifetime of the decal. As well we can disable all decals.

Globals

Variable Name	Description	Sample or Range
$pref::decalsOn	If set to true, decals are enabled, otherwise no decals will render.	[false , true]
$pref::Decal::maxNumDecals	Maximum decals allowed at any one time. Once this limit is breached, old decals start to be removed as new decals are added.	[0 , inf)
$pref::Decal::decalTimeout	Time in milliseconds it takes for a decal to be destroyed.	[0 , inf)

A.2.15. EditManager

While editing, it is possible to set bookmarks on the current position of the camera and then come back to that position by going back to the bookmark. Up to 10 bookmarks can be set. Setting a previously set bookmark will overwrite the old bookmark setting with a new bookmark setting. Bookmarks are not saved between sessions.

Console Method Summaries

setBookMark	gotoBookmark

Console Methods

gotoBookmark(slot)

Purpose
Use the ***gotoBookmark*** method to move the camera to the bookmark specified by slot.

Syntax
slot - An integer between 0 and 9.

Returns
No return value.

See Also
setBookmark

setBookmark(slot)

Purpose
Use the ***setBookmark*** method to set a bookmark on the current position of the camera. To bookmark is saved in a position specified by ***slot***.

Syntax
slot - An integer value between zero and nine.

Returns
No return value.

See Also
gotoBookmark

A.2.16. Explosion

The explosion class is used to represent various pyrotechnic displays. It is in fact a combination of multiple other classes used simultaneously or in sequence to produce various special effects.

A.2.17. ExplosionData

This is the data block that goes with the explosion class. It is used to describe all of the action be its of an individual explosion.

Fields

Field Name	Description	Sample or Range
camShakeAmp	The camera shake amplitude for all three axes: X, Y, and Z.	"1.0 2.0 3.0"
camShakeDuration	Time in seconds the shaking will occur over.	(-inf.0 , inf.0)
camShakeFalloff	Magnitude by which shaking decreases over distance to camShakeRadius.	(-inf.0 , inf.0)
camShakeFreq	The camera shake frequency for all three axes: X, Y, and Z.	"1.0 2.0 3.0"
camShakeRadius	Radius about the explosion in which shaking will be applied.	(-inf.0 , inf.0)
Debris	DebrisData datablock to use for this explosion.	see type
debrisNum	Total debris produced == debrisNum +/- debrisNumVariance.	(-inf , inf)
debrisNumVariance	Total debris produced == debrisNum +/- debrisNumVariance.	(-inf , inf)
debrisPhiMax	Maximum degrees of angular debris projection about the Y axis.	(-inf.0 , inf.0)
debrisPhiMin	Minimum degrees of angular debris projection about the Y axis.	(-inf.0 , inf.0)
debrisThetaMax	Maximum degrees of angular debris projection about an arbitray axis in the X-Z plane.	(-inf.0 , inf.0)
debrisThetaMin	Minimum degrees of angular debris projection about an arbitray axis in the X-Z plane.	(-inf.0 , inf.0)
debrisVelocity	Initial debris projection velocity == debrisVelocity +/- debrisVelocityVariance.	(-inf.0 , inf.0)
debrisVelocityVariance	Initial debris projection velocity == debrisVelocity +/- debrisVelocityVariance.	(-inf.0 , inf.0)
delayMS	Delay explosion 'effect' by delayMS +/- delayVariance after explosion object is created.	(-inf , inf)
delayVariance	Delay explosion 'effect' by delayMS +/- delayVariance after explosion object is created.	(-inf , inf)
emitter[0] emitter[1] emitter[2] emitter[3]	ParticleEmitterData to play when explosion occurs. Up to four are allowed.	see type
explosionscale	The scale of this explosion.	"1.0 2.0 3.0"
explosionShape	An optional shape that can be displayed at the time of the explosion with an (also) optional 'ambient' animation.	~/path/filename
faceViewer	Keep particles facing viewer.	[false , true]
lifetimeMS	Total life of explosion == lifetimeMS +/- lifetimeVariance.	(-inf , inf)
lifetimeVariance	Total life of explosion == lifetimeMS +/- lifetimeVariance.	(-inf , inf)
lightEndColor	Ending color of light (emitted by explosion).	"1.0 0.5 0.5"
lightEndRadius	Ending radius of light (emitted by explosion).	(-inf.0 , inf.0)

Field Name	Description	Sample or Range
lightStartColor	Starting color of light (emitted by explosion).	"1.0 0.5 0.5"
lightStartRadius	Starting radius of light (emitted by explosion).	(-inf.0 , inf.0)
offset	Explosion effects offset from explosion creation point.	(-inf.0 , inf.0)
particleDensity	Total number of particles to eject.	(-inf , inf)
particleEmitter	The emitter description (ParticleEmitterData) for this explosion.	see type
particleRadius	Maximum radius from effect center at which particles will be randomly created.	(-inf.0 , inf.0)
playSpeed	Overall rate of this explosion. Scales all effects.	(-inf.0 , inf.0)
shakeCamera	If set to true, enables camera shaking.	[false , true]
sizes[0] sizes[1] sizes[2] sizes[3]	Key-framing size controls for particles.	"1.0 2.0 3.0" "1.0 2.0 3.0" "1.0 2.0 3.0" "1.0 2.0 3.0"
soundProfile	description	see type
subExplosion[0] subExplosion[1] subExplosion[2] subExplosion[3] subExplosion[4]	Sub-explosions to play as part of this explosion. Up to five sub-explosions are allowed. Using parent explosions as sub-explosions or as children of sub-explosions will create an infinte loop of explosions.	ExplosionData datablock
times[0] times[1] times[2] times[3]	Key-framing time controls for particles.	[0.0, times[1]] [times[0], times[2]] [times[1], times[3]] [times[2], 1.0]

A.2.18. FileObject

Fileobject is a class provided to allow us to read write and modify files on the disk.

Console Method Summaries

close	**isEOF**	**openForAppend**
openForRead	**openForWrite**	**readLine**

Console Methods

close()

Purpose
Use the ***close*** method to close the current file handle. If the file was opened for writing, this flushes the contents of the last write to disk.

Returns
No return value.

See Also
openForAppend, openForRead, openForWrite

isEOF()

Purpose
Use the ***isEOF*** method to check to see if the end of the current file (opened for read) has been reached.

Returns
Returns true if the end of file has been reached, false otherwise.

See Also
openForRead

openForAppend(filename)

Purpose
Use the ***openForAppend*** method to open a previously created file for appending. If the file specified by filename does not exist, the file is created first.

Syntax
filename – The path and filename of the file to open for appending.

Returns
Returns true if the file was successfully opened for appending, false otherwise.

See Also
close, openForRead, openForWrite

openForRead(filename)and

Purpose
Use the ***openForRead*** method to open a previously created file for reading.

Syntax
filename – The path and filename of the file to open for reading.

Returns
Returns true if the file was successfully opened for reading, false otherwise.

See Also
close, OpenForAppend, OpenForWrite

openForWrite(filename)

Purpose
Use the ***openForWrite*** method to previously created or a new file for writing. In either case, the file will be overwritten.

Syntax
filename – The path and filename of the file to open for writing.

Returns
Returns true if the file was successfully opened for writing, false otherwise.

See Also
close, OpenForAppend, openForRead

readLine()

Purpose
Use the ***readLine*** method to read a single line from a file previously opened for reading.

Returns
Returns the next line in the file, or a NULL string if the end-of-file has been reached.

Notes
Use *isEOF* to check for end of file while reading.

See Also
isEOF, openForRead

writeLine(text)

Purpose
Use the ***writeLine*** method to write a value (***text***) into a file that was previously opened for appending or over-writing.

Syntax
text – The value to write to the file.

Returns
No return value.

See Also
openForAppend, openForWrite

A.2.19. FlyingVehicle

This class is used represent the flying variety of vehicles. It can be used to represent any variety of flying vehicle, including jets, biplanes, or rocket ships. If you do not actually need a flying vehicle, it is suggested that you use the hover vehicle instead which will closely approximate many similar needs.

Fields

Field Name	Description	Sample or Range
disableMove	If set to true, this vehicle will not be allowed to move.	[false , true]

A.2.20. FlyingVehicleData

Fields

Field Name	Description	Sample or Range
autoAngularForce	Angular stabilizer force. This force levels you out when autostabilizer kicks in.	[0.0 , 0.inf)
autoInputDamping	Amount by which input is damped to reduce oscillations caused by too much input.	[0.0 , 1.0)
autoLinearForce	Linear stabilzer force. This slows you down when autostabilizer kicks in.	[0.0 , 0.inf)
backwardJetEmitter	ParticleEmitterData datablock used for the thrust backward jet emitter. Emitter will be attached at node: JetNozzleX.	--
createHoverHeight	If set to true, uses hoverHeight to specify height to hover after creation.	[false , true]
downJetEmitter	ParticleEmitterData datablock used for the downward thrust jet emitter(s). Emitters will be attached at nodes: JetNozzle2 , and JetNozzle3.	--
engineSound	AudioProfile for engine ambient sound.	--
forwardJetEmitter	ParticleEmitterData datablock used for the forward thrust jet emitter(s). Emitters will be attached at nodes: JetNozzle0 , and JetNozzle1.	--
horizontalSurfaceForce	Wing thrust used in climbing and diving.	[0.0 , 0.inf)
hoverHeight	Height to hover at after creation if createHoverHeight is set to true.	[0.0 , 0.inf)
jetSound	AudioProfile for engine thrust sound.	--
maneuveringForce	Horizontal thrust force.	[0.0 , 0.inf)
maxAutoSpeed	Autostabilizer/AutoLinearForce are enabled when the vehicle's velocity magnitude drops below this value.	[0.0 , 0.inf)
minTrailSpeed	Minimum speed at which contrails turn on.	[0.0 , 0.inf)
rollForce	Automatically applied rolling force used to 'right' a tilted vehicle.	[0.0 , 0.inf)
rotationalDrag	Amount of drag applied to rotation about the vehicles up axis.	[0.0 , 0.inf)
steeringForce	Strength of turning thrust.	[0.0 , 0.inf)
steeringRollForce	How much you roll when turning.	[0.0 , 0.inf)

Field Name	Description	Sample or Range
trailEmitter	ParticleEmitterData datablock used for the contrail emitter(s). Emitters will be attached at nodes: contrail0 , contrail1, contrail2, and contrail3.	--
verticalSurfaceForce	Wing thrust used in 'sliding' side to side during turns.	[0.0 , 0.inf)
vertThrustMultiple	TypeF32	[0.0 , 0.inf)

A.2.21. fxFoliageReplicator

This class was originally intended to represent foliage in the scene. In fact, it can be used to represent anything that is displayed using a billboard. The value of this class is that all of the objects represented by a single foliage replicator are tied to a single instance of the class. Thus, regardless of the number of objects which may be displayed by any single foliage replicator the same network bandwidth is used. The billboards may be given a variety of behaviors and attributes. They may be stretched, self lighted, they may sway, and they may be are restricted to specific areas.

Fields

Field Name	Description	Sample or Range
Transform		
position	XYZ position of fx object.	"10.0 20. 0.0"
rotation	Values have no effect.	--
scale	Values have no effect.	--
Debugging		
useDebugInfo	Enable debug feedback. Some features must be uncommented in code to turn them on.	[false , true]
debugBoxHeight	Quad-tree box heights.	[0.0 , 0.inf)
HideFoliage	Stop displaying foliage.	[false , true]
showPlacementArea	Show the placement feedback device.	[false , true]
placementAreaHeight	Changes height of feedback device.	[0.0 , 0.inf)
placementColour	Changes color of feedback device.	"1.0 1.0 1.0 1.0"
Media/Replications		
seed	Value used to deterministically generate random object positions and parameters.	[0 , inf)
foliageFile	Texture file to use for foliage. Must be suitable for use as billboard.	~/path/filename
foliageCount	Number of billboards to replicate.	[0 , inf)
foliageRetries	Determines how many times to attempt to place a billboard. Retries are sometimes required in order to meet placement criteria. Failed placement attempts result fewer objects placed.	[0 , inf)
Area/Placement Radius		
innerRadiusX	X dimension of inner do-not-place ellipse. Objects are not allowed in ellipse described by this and the *InnerRadiusY* dimension.	[0.0 , 0.inf)
innerRadiusY	Y dimension of inner do-not-place ellipse. Objects are not allowed in ellipse described by this and the *InnerRadiusX* dimension.	[0.0 , 0.inf)

Field Name	Description	Sample or Range
outerRadiusX	X dimension of outer do-not-place ellipse. Objects are not allowed outside ellipse described by this and the *OuterRadiusY* dimension.	[0.0 , 0.inf)
outerRadiuxY	Y dimension of outer do-not-place ellipse. Objects are not allowed outside ellipse described by this and the *OuterRadiusX* dimension.	[0.0 , 0.inf)
Dimension		
minWidth	If scaling enabled (*FixSizeToMax* == false), this defines the billboard's minimum width.	[0.0 , 0.inf)
maxWidth	If scaling enabled (*FixSizeToMax* == false), this defines the billboard's maximum width.	[0.0 , 0.inf)
minHeight	If scaling enabled (*FixSizeToMax* == false), this defines the billboard's minimum height.	[0.0 , 0.inf)
maxHeight	If scaling enabled (*FixSizeToMax* == false), this defines the billboard's maximum height.	[0.0 , 0.inf)
fixAspectRatio	Prevents texture stretching.	[false , true]
fixSizeToMax	Forces all billboards to use *MaxWidth* and *MaxHeight* as their respective width and height.	[false , true]
offsetZ	This allows you to assist placement by lowering or raising the billboard by a fixed amount.	(-0.inf , 0.inf)
randomFlip	Allows random horizontal flipping of billboards, increasing variation for non-symmetric billboards.	[false , true]
Culling		
<useCulling>	Enables culling algorithms. i.e. test whether a billboard is within view before rendering it. **Warning:** Computing overhead offsets value for small placements. Only use for large placements if at all.	[false , true]
cullResolution	Determines size of culling quads. Lower values of *CullResolution* take longer to cull but more effectively removes non-visible billboards. Higher values of *CullResolution* take less time, but end up rendering more non-visible billboards. **Tradeoff:** Culling Time vs. Fill Rate.	[8 , inf)
viewDistance	When a billboards is at distance *ViewDistance* from the camera, it will be completely visible.	[0.0 , 0.inf)
viewClosest	Controls closest point at which billboards will begin to fade out.	[0.0 , 0.inf)
fadeInRegion fadeOutRegion	Together, these determine width of region between fade-in and fade-out.	[0.0 , 0.inf)
alphaCutoff	Controls general alpha level for rendering billboards.	[0.0 , 1.0]
groundAlpha	Controls alpha in region near ground. For fixing artifacts between billboard and ground.	[0.0 , 1.0]
Animation		
swayOn	Enables sway animation.	[false , true]
swaySync	All billboards sway in sync for this fx Object.	[false , true]
swayMagSide	Side-to-side swaying magnitude.	[0.0 , 0.inf)
swayMagFront	Back-and-forth swaying magnitude.	[0.0 , 0.inf)

Field Name	Description	Sample or Range
minSwayTime	Minimum sway time. Sway times are randomly chosen between a min and max on a per swing basis.	[0.0 , 0.inf)
maxSwayTime	Maximum sway time. Sway times are randomly chosen between a min and max on a per swing basis.	[0.0 , 0.inf)
Lighting		
lightOn	Turns on luminance (self lighting).	[false ,true]
lightSync	Light billboards in sync.	[false , true]
minLuminance	Minimum light value.	[0.0 , maxLuminance]
maxLuminance	Maximum light value.	[minLuminance , 1.0]
lightTime	Time required to transition from Min to Max and Max to Min. i.e. each duration is equal to *lightTime*.	[0.0 , 0.inf)
Restrictions/Restraints		
allowOnTerrain	Allows objects to be placed on terrain.	[false , true]
allowOnInteriors	Allows objects to be placed on interiors.	[false , true]
allowOnStatics	Allows objects to be placed on static shapes.	[false , true]
allowOnWater	Allows objects to be placed in area covered by water.	[false , true]
allowWaterSurface	Place on surface of water. Otherwise will be placed on terrain below water.	[false , true]
allowedTerrainSlope	Maximum slope to place on. Slopes beyond this value will be devoid of objects.	[false , true]

Console Functions

StartFoliageReplication()

startFoliageReplication()

Purpose
Use the ***startFoliageReplication*** function to start the foliage replication system.

Returns
No return value.

Notes
This can be called before be called after replication objects have been placed, but it should only be called once, and if it is not called, replicators will not work.

See Also
startClientReplication

A.2.22. fxLight

The fxLight object is used to represent in-game lights with and respective flares. This object casts (dynamic) light in the scene. It renders both a representation of the light source flare, and casts light on terrain and other objects.

Fields

Field Name	Description	Sample or Range
enable	Boolean value enabling this light.	[false , true]
iconSize	Floating-point value specifying render size of flare.	(-inf.0 , inf.0)

Console Method Summaries

attachToObject	**detachFromObject**	**reset**

Console Methods

`attachToObject( obj )`

Purpose
Use the ***attachToObject*** method to attach this fxLight object to GameBase object.

Syntax
obj - A GameBase derived object to attach to

Returns
No return value.

Notes
This method name is bit of a misnomer. When an fxLight object is attached to another object, it doesn't mean the fxLight will move with the object, but rather that if the object the fxLight is attached to is deleted, this object will be deleted. Also, the fxLight will always be processed (for rendering) after the object it is attached to.

See Also
detachFromObject

`detachFromObject()`

Purpose
Use the ***detachFromObject*** method to detach this fxLight from the last object it was attached to.

Returns
No return value.

See Also
attachToObject

reset()

Purpose
Use the ***reset*** method to lighting, animation, and texture of the current fxLight.

Returns
No return value.

setEnable(enabled)

Purpose
Use the ***setEnable*** method to enable or disable an fxLight.

Syntax
enabled – A boolean value. If set to true, this fxLight will cast light and be rendered.

Returns
No return value.

A.2.23. fxLightData

The data block associated with fxLight. This data block describes nearly all of the attributes of the fxLight object.Fields

Fields

Field Name	Description	Sample or Range
AnimBrightness	Enable brightness animation.	[false , true]
AnimColour	Enable color animation.	[false , true]
AnimOffsets	Enable offset animation.	[false , true]
AnimRadius	Enable radius animation.	[false , true]
AnimRotation	Enable rotation animation.	[false , true]
BlendMode	0 = GL_SRC_ALPHA, GL_ONE 1 = GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA 2 = GL_ONE, GL_ONE	[0 , 2]
BlueKeys	Key frame values for blue channel animation.	"ABCDABCD"
Brightness	Starting brightness.	[0.0 , 1.0]
BrightnessKeys	Key frame values for brightness animation.	"ABCDABCD"
BrightnessTime	Round-trip animation time for brightness.	[0.0 , 0.inf)
Colour	Initial light color.	"0.5 0.5 0.7 1.0"
ColourTime	Round-trip animation time for color.	[0.0 , 0.inf)
ConstantSize	Screen size for flare if not animated.	[0.0 , 0.inf)
ConstantSizeOn	Enable screen size animation for flare bitmap.	[false , true]
EndOffset	Ending offset for offset animations.	[0.0 , 0.inf)
FadeTime	Round-trip animation time for fade.	[0.0 , 0.inf)
FarDistance	Distance from light when flare achieves screen size of FarSize.	[0.0 , 0.inf)

Field Name	Description	Sample or Range
FarSize	Screen size for flare at FarDistance and beyond.	[0.0 , 0.inf)
FlareBitmap	Bitmap used represent the light source.	"~/data/flare.png"
FlareColour	Starting color for flare bitmap.	"0.5 0.5 0.7 1.0"
FlareOn	Enable flare bitmap.	[false , true]
FlareTP	Turns flare off in third person.	[false , true]
GreenKeys	Key frame values for green channel animation.	"ABCDABCD"
LerpBrightness	Enable brightness interpolation.	[false , true]
LerpColour	Enable color interpolation.	[false , true]
LerpOffset	Enable offset interpolation.	[false , true]
LerpRadius	Enable radius interpolation.	[false , true]
LerpRotation	Enable rotation interpolation.	[false , true]
LightOn	Enable light.	[false , true]
LinkFlare	Flag controlling whether the flare is linked to the light. If on then the flare tracks the color & brightness settings of the light to control its colorization.	[false , true]
LinkFlareSize	Flag controlling whether the flare is linked to the light by size. If on then the flare tracks the color & brightness settings of the light to control its size. The brighter the light, the larger the flare. The flare is scaled according to it's current animation e.g. it never gets any bigger than it would with this setting off.	[false , true]
MaxBrightness	Maximum brightness when brightness animation is enabled.	[0.0 , 0.inf)
MaxColour	Maximum color when color animation is enabled.	"0.5 0.5 0.7 1.0"
MaxRadius	Maximum radius when radius animation is enabled.	[0.0 , 0.inf)
MaxRotation	Maximum rotation when rotation animation is enabled.	[0.0 , 0.inf)
MinBrightness	Minimum brightness when brightness animation is enabled.	[0.0 , 0.inf)
MinColour	Minimum color when color animation is enabled.	"0.5 0.5 0.7 1.0"
MinRadius	Minimum radius when radius animation is enabled.	[0.0 , 0.inf)
MinRotation	Minimum rotation when rotation animation is enabled.	[0.0 , 0.inf)
NearDistance	Distance from light when flare achieves screen size of NearSize.	[0.0 , 0.inf)
NearSize	Screen size for flare at NearDistance and closer.	[0.0 , 0.inf)
OffsetKeys	Key frame values for offset animation.	"ABCDABCD"
OffsetTime	Round-trip animation time for offset.	[0.0 , 0.inf)
Radius	Initial light sphere radius.	[0.0 , 0.inf)
RadiusKeys	Key frame values for radius animation.	"ABCDABCD"
RadiusTime	Round-trip animation time for radius.	[0.0 , 0.inf)
RedKeys	Key frame values for red channel animation.	"ABCDABCD"
RotationKeys	Key frame values for rotation animation.	"ABCDABCD"
RotationTime	Round-trip animation time for rotation.	[0.0 , 0.inf)
SingleColourKeys	Use RedKeys to animate all colors at the same time.	"ABCDABCD"
StartOffset	Starting offset for offset animation.	[0.0 , 0.inf)

A.2.24. fxShapeReplicator

This class is used to replicate any shape. It works in nearly the same way as the fxFoliageReplicator. Each of the individual shapes may enable or disable a collision box allowing them to be interacted with or ignore it in collision interactions.

Fields

Field Name	Description	Sample or Range
Transform		
position	XYZ position of fx object.	"10.0 20. 0.0"
rotation	Values have no effect.	--
scale	Values have no effect.	--
Debugging		
HideReplications	Stop displaying shapes.	[false , true]
ShowPlacementArea	Show the placement feedback device.	[false , true]
PlacementAreaHeight	Changes height of feedback device.	[0.0 , inf.0)
PlacementColour	Changes color of feedback device.	"1.0 0.5 0.5 1.0"
Media/Replications		
Seed	Value used to deterministically generate random object positions and parameters.	[0 , inf)
shapeFile	DTS file to replicate.	"~/data/myshape.dts"
ShapeCount	Number of shapes to replicate.	[0 , inf)
ShapeRetries	Determines how many times to attempt to place a shape. Retries are sometimes required in order to meet placement criteria. Failed placement attempts result fewer objects placed.	[0 , inf)
Area/Placement Radius		
InnerRadiusX	X dimension of inner do-not-place ellipse. Objects are not allowed in ellipse described by this and the *InnerRadiusY* dimension.	[0.0 , 0.inf)
InnerRadiusY	Y dimension of inner do-not-place ellipse. Objects are not allowed in ellipse described by this and the *InnerRadiusX* dimension.	[0.0 , 0.inf)
OuterRadiusX	X dimension of outer do-not-place ellipse. Objects are not allowed outside ellipse described by this and the *OuterRadiusY* dimension.	[0.0 , 0.inf)
OuterRadiuxY	Y dimension of outer do-not-place ellipse. Objects are not allowed outside ellipse described by this and the *OuterRadiusX* dimension.	[0.0 , 0.inf)
Restrictions/Restraints		
AllowOnTerrain	Allows objects to be placed on terrain.	[false , true]
AllowOnInteriors	Allows objects to be placed on interiors.	[false , true]
AllowOnStatics	Allows objects to be placed on static shapes.	[false , true]
AllowOnWater	Allows objects to be placed in area covered by water.	[false , true]
AllowWaterSurface	Place on surface of water. Otherwise will be placed on terrain below water.	[false , true]
AllowedTerrainSlope	Maximum slope to place on. Slopes beyond this value will be devoid of objects.	[false , true]

Field Name	Description	Sample or Range
AlignToTerrain	Causes DTS shapes to align to up vector of terrain if placed on terrain.	[false , true]
Interactions	Enables collision boxes if DTS shape has one.	[false , true]
TerrainAlignment	Vector to adjust how shape aligns to terrain when *AlighnToTerrain* is true.	"0.7 0.7 0.7"
Object Transforms		
ShapeScaleMin	Minimum randomly selected scale for DTS shape.	[0.0 , 0.inf)
ShapeScaleMax	Maximum randomly selected scale for DTS shape.	[0.0 , 0.inf)
ShapeRotationMin	Minimum random rotation for DTS shape.	[0.0 , 0.inf)
ShapeRotationMax	Maximum random rotation for DTS shape.	[0.0 , 0.inf)
OffsetZ	This allows you to assist placement by lowering or raising the shape by a fixed amount.	" 0.0 1.2 0.0 "

Console Functions

StartClientReplication()

StartClientReplication()

Purpose
Use the ***StartClientReplication*** function to start the shape replication system.

Returns
No return value.

Notes
This can be called before be called after replication objects have been placed, but it should only be called once, and if it is not called, replicators will not work.

See Also
startFoliageReplication

A.2.25. fxSunLight

This class is used to represent celestial bodies. You may defined as many fxSunlight objects as you need in your seeing. Each object may be animated independently. As a whole, this class provides several features allowing us to represent a variety of celestial bodies with many different behaviors.

Fields

Field Name	Description	Sample or Range
AnimAzimuth	Enable azimuth animation.	[false , true]
AnimBrightness	Enable brightness animation.	[false , true]
AnimColour	Enable color animation.	[false , true]
AnimElevation	Enable elevation animation.	[false , true]
AnimRotation	Enable rotation animation.	[false , true]
AnimSize	Enable size animation.	[false , true]
AzimuthKeys	Key frame values for azimuth animation.	"ABCDABCD"
AzimuthTime	Round-trip animation time for azimuth.	[0.0 , inf.0)
BlendMode	0 = GL_SRC_ALPHA, GL_ONE 1 = GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA 2 = GL_ONE, GL_ONE	[0 , 2]
BlueKeys	Key frame values for blue channel animation.	"ABCDABCD"
Brightness	Starting brightness.	[0.0 , inf.0)
BrightnessKeys	Key frame values for brightness animation.	"ABCDABCD"
BrightnessTime	Round-trip animation time for brightness.	[0.0 , inf.0)
Colour	Initial color.	"1.0 1.0 0.0 1.0"
ColourTime	Round-trip animation time for color.	[0.0 , inf.0)
ElevationKeys	Key frame values for elevation animation.	"ABCDABCD"
ElevationTime	Round-trip animation time for elevation.	[0.0 , inf.0)
Enable	Enable or disable fxSunlight object rendering.	[false , true]
FadeTime	Round-trip animation time for fade.	[0.0 , inf.0)
FlareSize	Size for flare bitmap.	[0.0 , inf.0)
FlareTP	If true, enables flare in third person.	[false , true]
GreenKeys	Key frame values for green channel animation.	"ABCDABCD"
LerpAzimuth	Enable azimuth interpolation.	[false , true]
LerpBrightness	Enable brightness interpolation.	[false , true]
LerpColour	Enable color interpolation.	[false , true]
LerpElevation	Enable elevation interpolation.	[false , true]
LerpRotation	Enable rotation interpolation.	[false , true]
LerpSize	Enable size interpolation.	[false , true]
LinkFlareSize	Flag controlling whether the flare is linked to the light by size. If on then the flare tracks the color & brightness settings of the light to control its size. The brighter the light, the larger the flare. The flare is scaled according to it's current animation e.g. it never gets any bigger than it would with this setting off.	[false , true]
LocalFlareBitmap	Lens flare texture.	~/path/filename.png

Field Name	Description	Sample or Range
LockToRealSun	Elevation and azimuth for this object follows elevation and azimuth for Sun object.	[false , true]
MaxAzimuth	Maximum azimuth setting.	[0.0 , 360.0)
MaxBrightness	Maximum brightness when brightness animation is enabled.	[0.0 , inf.0)
MaxColour	Maximum color when color animation is enabled.	
MaxElevation	Maximum elevation.	[-90.0 , 90.0]
MaxRotation	Maximum rotation when rotation animation is enabled.	[0.0 , 360.0)
MaxSize	Maximum size for flare.	[0.0 , inf.0)
MinAzimuth	Minimum azimuth setting.	[0.0 , 360.0)
MinBrightness	Minimum brightness when brightness animation is enabled.	[0.0 , inf.0)
MinColour	Minimum color when color animation is enabled.	"0.0 0.0 1.0 1.0"
MinElevation	Minimum elevation setting.	[-90.0 90.0]
MinRotation	Minimum rotation when rotation animation is enabled.	[0.0 , 360.0)
MinSize	Minimum flare size.	[0.0 , inf.0)
NearDistance	Distance from light when flare achieves screen size of NearSize.	[0.0 , inf.0)
NearSize	Screen size for flare at NearDistance and closer.	[0.0 , inf.0)
OffsetKeys	Key frame values for offset animation.	"ABCDABCD"
OffsetTime	Round-trip animation time for offset.	[0.0 , inf.0)
RedKeys	Key frame values for red channel animation.	"ABCDABCD"
RemoteFlareBitmap	The sun texture.	~/path/filename.png
RotationKeys	Key frame values for rotation animation.	"ABCDABCD"
RotationTime	Round-trip animation time for rotation.	[0.0 , inf.0)
SingleColourKeys	Use RedKeys to animate all colors at the same time.	"ABCDABCD"
SizeKeys	Key frame values for size animation.	"ABCDABCD"
SizeTime	Round-trip animation time for size.	[0.0 , inf.0)
SunAzimuth	Initial azimuth setting.	[0.0 , 360.0)
SunElevation	Initial elevation setting.	[-90.0 , 90.0]

Console Method Summaries

Long descriptions for these functions have not been supplied because the functions are very simple and their names are quite descriptive.

reset()	**setAzimuthKeys(keys)**	**setAzimuthTime(time)**
setBlendMode(mode)	**setBlueKeys(keys)**	**setBrightnessKeys(keys)**
setBrightnessTime(time)	**setColourTime(time)**	**setElevationKeys(keys)**
setElevationTime(time)	**setEnable(status)**	**setFadeTime(time)**
setFlareBitmaps(local , remote)	**setFlareBrightness(brightness)**	**setFlareColour(r , g , b)**
setFlareSize(size)	**setFlareTP(status)**	**setGreenKeys(keys)**
setLerpAzimuth(status)	**setLerpBrightness(status)**	**setLerpColour(status)**
setLerpElevation(status)	**setLerpRotation(status)**	**setLerpSize(status)**
setLinkFlareSize(status)	**setMaxAzimuth(azimuth)**	**setMaxBrightness(brightness)**
setMaxColour(r , g , b)	**setMaxElevation(elevation)**	**setMaxRotation(rotation)**
setMaxSize(size)	**setMinAzimuth(azimuth)**	**setMinBrightness(brightness)**

reset()	**setAzimuthKeys(keys)**	**setAzimuthTime(time)**
setMinColour(r , g , b)	**setMinElevation(elevation)**	**setMinRotation(rotation)**
setMinSize(size)	**setRedKeys(keys)**	**setRotationKeys(keys)**
setRotationTime(time)	**setSingleColourKeys(status)**	**setSizeKeys(keys)**
setSizeTime(time)	**setSunAzimuth(azimuth)**	**setSunElevation(elevation)**
setUseAzimuth(status)	**setUseBrightness(status)**	**setUseColour(status)**
setUseElevation(status)	**setUseRotation(status)**	**setUseSize(status)**

A.2.26. GameBase

This is the base class to all classes using or requiring a data block. This class can be considered virtual as you do not create instances of it in the game world.

Fields

Field Name	Description	Sample or Range
dataBlock	The corresponding data block for this object.	GameBaseData datablock

Globals

Variable Name	Description	Sample or Range
$gameBase::boundingBox	If true, objects will display their bounding boxes.	[false , true]

Console Method Summaries

getDataBlock	**setDataBlock**

Console Methods

getDataBlock()

Purpose
Use the ***getDataBlock*** method to get the datablock used by this object.

Returns
No return value.

See Also
setDataBlock

setDataBlock(db)

Purpose
Use the ***setDataBlock*** method to change the datablock for this object to a new datablock represented by db.

Syntax
db – A datablock name or ID.

Returns
Returns true if the datablock with successfully changed, otherwise it returns false.

Notes
The new datablock must be of a compatible type with the current class.

See Also
getDataBlock

A.2.27. GameBaseData

This class is the corresponding data block the gamebase. It can be considered virtual also.

Fields

Field Name	Description	Sample or Range
category	Creator category to place this object in.	"MyCategory"
className	A new namespace to be inserted between the name of the data block and the class name of the data block.	"SomeClassname"

A.2.28. GameConnection

A class connecting the client in the current executable to a server in the current executable or to a server and external executable.

Console Method Summaries

activateGhosting	chaseCam	clearCameraObject
delete	getCameraObject	getControlCameraFov
getControlObject	getServerConnection	isAIControlled
isDemoPlaying	isDemoRecording	isFirstPerson
listClassIDs	play2D	play3D
playDemo	resetGhosting	setBlackOut
setCameraObject	setConnectArgs	setControlCameraFov
setControlObject	setFirstPerson	setJoinPassword
setMissionCRC	startRecording	stopRecording

Console Methods

activateGhosting()

Purpose
Use the ***activateGhosting*** method to GameConnection instance to start ghosting objects to the client.

Returns
No return value.

Notes
This is called on each client connection by the server.

See Also
resetGhosting

chaseCam(delay)

Purpose
Use the ***chaseCam*** method to insert a delay in between translations and rotations of the control object and the attached 3rd POV camera. That is, if ***delay*** is a positive non-zero value, the camera will lag behind translations and rotations of the 3rd POV player.

Syntax
delay – An integer value equal to or greater than zero. If this value is greater than zero, the camera will begin to lag the players translations and rotations.

Notes
This can quickly become disconcerting as it disconnects player input from the results. Use this feature cautiously.

delete([reason])

Purpose
Use the ***delete*** method to destroy and disconnect the current connection, giving an optional ***reason***. If reason is specified, it will be transmitted to the client/server on the other end of the connection.

Syntax
reason – A string explaining while the connection is being severed.

Returns
No return value.

getCameraObject()

Purpose
Use the ***getCameraObject*** method to get the current camera object ID.

Returns
Returns the ID of the current camera object, or the control object if the POV is first person.

See Also
setCameraObject()

getControlCameraFov()

Purpose
Use the ***getControlCameraFov*** method to FOV of the current control 'camera'.

Returns
Returns a FOV angle between 0.0 and 180.0.

Notes
This may not actually be the FOV of the camera object, but rather the FOV being used by the camera. The difference is subtle, but if a camera is attached to an object and using that object's FOV setting, then the FOV is actually coming from the object, not the camera.

See Also
getCameraObject, setControlCameraFov

getControlObject()

Purpose
Use the ***getControlObject*** method to get the ID of the current control-object for this GameConnection.

Returns
Returns an integer value representing the ID of an object acting as the control-object for this GameConnection. If no control-object was set up, this will return 0.

getServerConnection()

Purpose
Use the ***getServerConnection*** method to get the ID of the server this client-side GameConnection is attached to.

Returns
Returns an integer representing the ID of a server-side GameConnection on the other end of this client-side GameConnection. If no server is connected, or if this GameConnection is a server-side connection, the return value is 0.

isAIControlled()

Purpose
Use the ***isAIControlled*** method to determine if this is an AIGameConnection.

Returns
Returns true if this is an AIGameConnection instance.

Notes
This will always return false for a GameConnection and always return true for an AIGameConnection. It was defined here to allow scripts to call it universally on either type of connection.

isDemoPlaying()

Purpose
Use the ***isDemoPlaying*** method to see if this GameConnection is playing a demo.

Returns
Returns true if a demo is being played, false otherwise.

See Also
isDemoRecording, playDemo, startRecording, stopRecording

isDemoRecording()

Purpose
Use the ***isDemoRecording*** method to see if this GameConnection is recording a demo.

Returns
Returns true if a demo is being recorded, false otherwise.

See Also
isDemoPlaying, playDemo, startRecording, stopRecording

isFirstPerson()

Purpose
Use the ***isFirstPerson*** method to see if the camera attached to this GameConnection is in 1st POV.

Returns
Returns true if the camera attached to this GameConnection is in 1st POV, false otherwise.

See Also
setFirstPerson

listClassIDs()

Purpose
Use the ***listClassIDs*** method to dump a list of all class IDs that this GameConnection knows about.

Returns
No return value.

Notes
This is a debug feature.

play2D(AP)

Purpose
Use the ***play2D*** method to play a 2D AudioProfile previously specified using the datablock keyword.

Syntax
AP – A 2D AudioProfile previously specified using the datablock keyword.

Returns
Returns true if the sound can be played, false otherwise.

Notes
Be sure to use only AudioProfiles created using the datablock keyword.

See Also
play3D

play3D(AP , position)

Purpose
Use the ***play2D*** method to play a 3D AudioProfile previously specified using the datablock keyword.

Syntax
AP – A 3D AudioProfile previously specified using the datablock keyword.
position – The position to play this sound at.

Returns
Returns true if the sound can be played, false otherwise.

Notes
Be sure to use only AudioProfiles created using the datablock keyword.

See Also
play2D

playDemo(demoFileName)

Purpose
Use the ***playDemo*** method to play a previously recorded demo on this GameConnection.

Syntax
demoFileName – A path and file name pointing to a previously recorded demo.

Returns
Returns true if the demo was successfully played, false otherwise.

See Also
isDemoPlaying, isDemoRecording, startRecording, stopRecording

resetGhosting()

Purpose
Use the ***resetGhosting*** method to reset ghosting. This in effect tells the server to resend each ghost to insure that all objects which should be ghosts and are in fact ghosted.

Returns
No return value.

See Also
activateGhosting

setBlackOut(doFade , timeMS)

Purpose
Use the ***setBlackOut*** method to fade (black) the screen out, or in.

Syntax
doFade – A boolean value. If set to true, the screen is blacked out, if false, the blackout is reversed.
timeMS – An integer value specifying the time it takes to fade-out or to fade back in.

Returns
No return value.

Notes
This is only called on the client-side GameConnection (from client to server), calling in on a server-side connection (from server to client), will do nothing.

The GameBase class provides two similar features for whiting out the screen and causing damage flashes.

setCameraObject(newCamera)

Purpose
Use the ***setCameraObject*** method to change the camera object associated with this GameConnection to ***newCamera***.

Syntax
newCamera – The ID of a valid camera object.

Returns
Returns true on success and false on failure.

Notes
This can only be called on the serer-side GameConnection.

See Also
getCameraObject

setConnectArgs(name [, arg1 , ... , arg15])

Purpose
Use the ***setConnectArgs*** method to set the connection arguments for this client-side GameConnection. These values will be passed to the server upon establishing a connection.

Syntax
name – Generally, the first argument is the name of the player.
arg1 , ... , arg15 – 15 additional arguments may be passed.

Returns
No return value.

See Also
setJoinPassword

setControlCameraFov(newFOV)

Purpose
Use the ***setControlCameraFov*** method to change the FOV of the control object 'camera'.

Syntax
newFOV – A floating-point value between 0.0 and 180.0 representing the new field-of-view setting for any and all cameras attached to this connection.

Returns
No return value.

Notes
This setting is sticky. That is, as the camera object is changed, this value will be retained. These values are however constrained by the datablock min and max FOV settings that are currently in effect.

See Also
getControlCameraFov

setControlObject(object)

Purpose
Use the ***setControlObject*** method to change the control object for this server-side GameConnection to the GameBase object specified by ***object***.

Syntax
object – A valid GameBase object to make the new control object.

Returns
Returns true if the control object was successfully changed, otherwise returns false.

Notes
The same control object may NOT be used by more than one client connection.

See Also
getControlObject

setFirstPerson(isFirstPerson)

Purpose
Use the ***setFirstPerson*** method to change the current point-of-view to first or third-person, depending on the value of ***isFirstPerson***.

Syntax
isFirstPerson – A boolean value. If set to true, the POV for this GameConnection is set to 1st POV, otherwise it is set to 3rd POV.

Returns
No return value.

Notes
This will NOT override value specified in the control-object/camera datablocks enabling or disabling a particular POV.

See Also
isFirstPerson

setJoinPassword(password)

Purpose
Use the ***setJoinPassword*** method to set the ***password*** required to connect to this server-side GameConnection.

Syntax
password – A string representing the case insensitive password to use for this server-side GameConnection.

Returns
No return value.

Notes
Pass a NULL string to clear the password.

See Also
setConnectArgs

setMissionCRC(CRC)

Purpose
Use the ***setMissionCRC*** method to set the cyclic-redundancy-check (CRC) value for the current mission. This allows clients to determine whether they need to relight a scene or not. The CRC value of a mission file is used to calculate the signature of a mission lighting file (.ml). If the signature does not match the current CRC, the mission will be relit.

Syntax
CRC – An integer value representing the cyclic-redundancy-check code for the current misssion file.

Returns
No return value.

startRecording(fileName)

Purpose
Use the ***startRecording*** method to tell the GameConnection to start recording a demo. The demo will be stored at the location specified by ***fileName***.

Syntax
fileName - To location to store the demo file. This should be a complete filename, including path and file name.

Returns
No return value.

See Also
isDemoPlaying, isDemoRecording, playDemo, stopRecording

stopRecording()

Purpose
Use the ***stopRecording*** method to tell the GameConnection to stop recording a demo.

Returns
No return value.

See Also
isDemoPlaying, isDemoRecording, playDemo, startRecording

transmitDataBlocks(sequence)

Purpose
Use the ***transmitDataBlocks*** method to start sending datablocks to the client connected to this GameConnection.

Syntax
sequence - A book-keeping value used by scripts.

Returns
No return value.

A.2.29. HoverVehicle

This class is used represent the hover variety of vehicles. Space A hover vehicle is any vehicle that must maintain a specific distance between the vehicle and the ground at all times, which is not the same as a flying vehicle which may change this distance.

A.2.30. HoverVehicleData

The datablock associated with hover vehicle class. This class describes all of the behaviors of the hover vehicle.

Fields

Field Name	Description	Sample or Range
brakingActivationSpeed	Minimum rate at which braking force will be activated.	[0.0 , inf.0)
brakingForce	A force used to bring a drifting hover vehicle to a halt.	[0.0 , inf.0)
dragForce	A percentage of the current velocity applied in the opposite direction of the hover vehicles current motion.	[0.01 , 1]
dustTrailEmitter	ParticleEmitterData datablock for dust trails.	ParticleEmitterData datablock
dustTrailFreqMod	Dust trail modulator. Lower values equal more dense trail, higher values equal more sparse trail.	(0.0 , inf.0)
dustTrailOffset	A vector by which to offset the dust trail emitter from it's mount node. By default the emitter plays directly beneath the center of the vehicle.	"0.0 5.0 0.0"
engineSound	Engine sound special effect.	AudioProfile datablock
floatingGravMag	Gravity modifier that is enabled while vehicle is in contact with other objects.	[0.0 , 1.0]
floatingThrustFactor	When vehicle is NOT floatingFloating thrust is equal to: (floatingThrustFactor * mainThrustForce) , otherwise it is 100% of mainThrustForce.	[0.0 , 1.0]
floatSound	Floating sound effect.	AudioProfile datablock
forwardJetEmitter	ParticleEmitterData datablock used for forward thrust jets, attached at JetNozzle0 and JetNozzle1.	ParticleEmitterData datablock
gyroDrag	Gyroscopic force that resists tilting.	[0.0 , inf.0)
jetSound	Jetting sound special effect.	AudioProfile datablock
mainThrustForce	Forward thrust factor.	[0.0 , inf.0)
normalForce	A force that attempts to right a tilted vehicle.	[0.0 , inf.0)
pitchForce	A force applied forward or backward, depending on the amount of vehicle pitch.	[0.0 , inf.0)
restorativeForce	Another minor force used to level a tilted vehicle.	[0.0 , inf.0)
reverseThrustForce	Reverse thrust factor.	[0.0 , inf.0)
rollForce	A force applied left or right, depending on the amount of vehicle roll.	[0.0 , inf.0)
stabDampingConstant	A value used to keep the vehicle from bouncing up-and down with minor elevation changes.	[0.0 , 1.0]
stabLenMax	A limiting factor applied to the automatically generated stabilizing 'box'. This box grows and shrinks as the velocity of the hover vehicle increase and decreases. This phantom box will collide with objects so limiting it's maximum size is a good idea.	[0.0 , inf.0)
stabLenMin	A limiting factor applied to the automatically generated stabilizing 'box'. This box grows and shrinks as the velocity of the hover vehicle increase and decreases. If the box becomes too small, the vehicle will become unstable and prone to tilting.	[0.0 , inf.0)

Field Name	Description	Sample or Range
stabSpringConstant	This value should be at least twice the mass of the hoverVehicle and is used to keep the vehicle from sinking to the ground.	[0.0 , inf.0)
steeringForce	Controls strength of steering controls.	[0.0 , inf.0)
strafeThrustForce	Side to side thrust strength.	[0.0 , inf.0)
triggerTrailHeight	Height at which dust trail emitter is enabled.	[0.0 , inf.0)
turboFactor	A multiplier applied to thrustforce while jetting ($mvTriggerCount3 > 0).	[0.0 , inf.0)
vertFactor	A multiplier to use to increase or decrease drag in the vertical direction.	[0.0 , inf.0)

A.2.31. InteriorInstance

This class is used represent interiors. That is, objects/meshes use to represent buildings bridges and other large objects in the world which can be entered. As the name would imply, interiors have an inside, and interior. Interior instance utilizes a BSP collision algorithm, as well as portals and other features.

Fields

Field Name	Description	Sample or Range
interiorFile	The DIF filename or for this interior.	~/path/filename.dif
showTerrainInside	If false, and interior uses portals, terrain will not render inside the interior.	[false , true]
useGLLighting	If this is set to true, interiors wil have basic GL lighting as soon as they are placed, otherwise a relight will be required to make textures show up.	[false , true]

Globals

Variable Name	Description	Sample or Range
pref::Interior::LightUpdatePeriod	Millisecond between dynamic light updates for interiors. Default is 66 milliseconds.	[1 , inf]
pref::Interior::ShowEnvironmentMaps	Enables environmental mapping for all interiors.	[false , true]
pref::Interior::DynamicLights	Enable dynamic lights for all interiors.	[false , true]
pref::Interior::VertexLighting	Enable vertex lighting for all interiors.	[false , true]
pref::Interior::TexturedFog	Enables advanced fog rendering for all interiors.	[false , true]
pref::Interior::lockArrays	Advanced rendering feature.	[false , true]
pref::Interior::detailAdjust	LOD transition tolerance.	[0.3 , 1.0]
pref::Interior::DontRestrictOutside	Debug mode.	[false , true]

Console Method Summaries

activateLight	deactivateLight	echoTriggerableLights
getNumDetailLevels	magicButton	setAlarmMode

Console Methods

activateLight(lightName)

Purpose
Use the ***activateLight*** method to enable a named light in the current interior.

Syntax
lightName – A string containing the name of a light to be activated.

Returns
No return value.

See Also
deactivateLight, echoTriggerableLights

deactivateLight(lightName)

Purpose
Use the ***deactivateLight*** method to disable a named light in the current interior.

Syntax
lightName – A string containing the name of a light to be deactivated.

Returns
No return value.

See Also
activateLight, echoTriggerableLights

echoTriggerableLights()

Purpose
Use the ***echoTriggerableLights*** method to dump a list of all the triggerable lights this interior contains to the console.

Returns
No return value.

Notes
This is a nice helper function for those of us who can't always remember our light names.

See Also
activateLight, deactivateLight

getNumDetailLevels()

Purpose
Use the ***getNumDetailLevels*** method to determine the number of details that the current interior has.

Returns
Returns an integer value representing the maximum number of detail levels present in this interior.

Notes
Detail levels start from 0 and increment, where 0 has the most detail, 1 less than 0, 2 less than 1, et cetera.

See Also
setDetailLevel

setAlarmMode(mode)

Purpose
Use the ***setAlarmMode*** method to set the global alarm mode for this interior. This will enable or disable all alarm mode lights.

Syntax
mode – A string. If this value is set to "On", then all alarm mode lights will be enabled. Any other value disables all alarm mode lights.

Returns
No return value.

setDetailLevel(level)

Purpose
Use the ***setDetailLevel*** method to force the current interior to render at a specific LOD, if the interior supports that ***level***.

Syntax
level – An integer value greater than or equal to zero, representing a LOD to render this interior at.

Returns
No return value.

Notes
Detail levels start from 0 and increment, where 0 has the most detail, 1 less than 0, 2 less than 1, et cetera.

See Also
getNumDetailLevels

setSkinBase(basename)

Purpose
Use the ***setSkinBase*** method to switch skins on an interior.

Syntax
basename – Text string equivalent to new texture(s) prefix.

Notes
Currently not working.

A.2.32. Item

This class, derived from ShapeBase, is used to represent small object in the world which are intended to be interacted with and/or picked up. That is coinscoins, grenades, rifles, gems, etc. This class of object does not require a collision mass although having one is useful in certain circumstances, specifically for line of sight collisions. By default the engine will supply a simple collision mesh for object to object collisions. This object supplies a several physical as well as visible behaviors, including object specific gravity, elasticity, lighting, etc.

Fields

Field Name	Description	Sample or Range
collidable	--	deprecated
rotate	If set to a true, this item will auto-rotate.	[false , true]
static	if this value is set to true, the item will not move when placed. Conversely, a non-static shape is allowed to move.	[false , true]

Console Method Summaries

getLastStickyNormal	**getLastStickyPos**	**isRotating**

Console Methods

getLastStickyNormal()

Purpose
Use the ***getLastStickyNormal*** method to get the normal vector for the last sticky collision. If the item is not set to sticky, that is the datablock field *sticky* is set to false, this object will not track collisions.

Returns
Returns a string containing the XYZ values of the normal vector. If no normal vector is available, or if the item is not sticky this method will return the NULL string.

Notes
To enable sticky collisions, set the *ItemData.sticky* field to true.

See Also
getLastStickyPos

getLastStickyPos()

Purpose
Use the ***getLastStickyPos*** method to get the position vector for the last sticky collision. If the item is not set to sticky, that is the datablock field *sticky* is set to false, this object will not track collisions.

Returns
Returns a string containing the XYZ coordinates of the last collision position. If no normal vector is available, or if the item is not sticky this method will return the null string ("").

Notes
To enable sticky collisions, set the *ItemData.sticky* field to true.

See Also
getLastStickyNormal

isRotating()

Purpose
Use the ***isRotating*** method to determine if this item is playing the built-in rotation animation.

Returns
Returns true if the current item is rotating, that is if the rotate field was set to true when this object was created.

Notes
To enable rotation, set the *rotate* field in the item instance to true.

isStatic()

Purpose
Use the ***isStatic*** method to determine if this is a static item. A static item will not change position as the result of the application of outside forces like gravity and impulses.

Returns
Returns true if the current item is static, that is the static field was set to true when this object was created.

Notes
To make an item static, set the *static* field in the item instance to true.
A static item can still play animations and auto-rotate.

See Also
isRotating

setCollisionTimeout(obj)

Purpose
Use the ***setCollisionTimeout*** method to disable collisions between a specific GameBase derived object and this item. The default period for this timeout is 15 ticks (15 x 32ms == 480ms) or roughly one-half second.

Syntax
obj – The name or ID of a GameBase derived object with which to ignore collisions for ~1/2 second.

Returns
Returns true if the collision timeout was successfully set.

Notes
To modify the default timeout duration, see *sCollisionTimeout* in item.cc. It should be noted, that the colliding object is the one that is told to (temporarily) ignore subsequent collisions, not the item.

A.2.33. ItemData

This is the datablock associated with the item object. This class specifies most of the behavior of an item.

Fields

Field Name	Description	Sample or Range
Pickups		
pickupName	Message to be printed when this object is picked up. It is the responsibility of scripts to print this message.	--
Physics		
elasticity	A floating-point value specifying how 'bouncy' this object is. i.e. How much of the shape's velocity is lost onCollision(). **Note:** Because of rounding errors between gravity and elasticity calculations, an object with 1.0 elasticity that is bouncing up and down will eventually bounce away.	Allowed: [0.0 , inf) Suggested: [0.0 , 1.0)
friction	A floating-point value specifying how much velocity is lost to impact and sliding friction.	Allowed:(-inf , inf) Suggested: [0.0 , 1.0]

gravityMod	A floating-point value specifying an individual variance of local gravity for this item. In other words, an item can be told to ignore gravity, or to be affected by 2x gravity, or even to float upward.	Allowed:(-inf , inf)
		Suggested:(-20.0 , 20.0)

maxVelocity	A floating-point value specifying a limit on this item's maximum velocity. This can be used to keep the item from falling for flying too fast. It is also helpful to limit the effects of other settings like velocity, etc.	
sticky	A boolean value specifying whether this object will arrest (stop) on impact. Furthermore, if an object is sticky, it retains information about the last location is hit which can be accessed via console methods (see below).	[true, false]
Type		
dynamicType	An integer value which, if specified, is added to the value returned by getType().	See dynamicType below
Light		
lightColor	A three-value floating-point vector containing the RGB components of this item's light.	Individually: [0.0 , 1.0]
lightOnlyStatic	A boolean value instructing the item to produce light, only in the case that the object field 'isStatic' is true.	[false , true]
lightRadius	A floating point value specifying the radius for this item's light.	[0.0 , 20.0]
lightTime	A floating point value specifying the time (in milliseconds) it takes for a pulsing light to transition on-off-on.	(100 , inf)
lightType	A string specifying the type of light, if any, this shape emits.	"NoLight" "ConstantLight" "PulsingLight"

A.2.34. Lightning

This class is used to represent the special effect lightning. Lightning can be generated or rendered from a bitmap. Lightning bolts start at the top of the specified lightning block and terminate somewhere at the bottom or below the specified lightning block.

Fields

Field Name	Description	Sample or Range
boltStartRadius	This field specifies the radius within which lightning bolts will start.	(0.0 , inf)
chanceToHitTarget	Defunct.	[0.0 , 1.0]
color	This field is used to specify the color of lightning when it is created. Note, this is the exact color for generated lightning but should be considered a filter color for lightning that he uses bitmaps	"r g b a" (float)
fadeColor	This field is used to specify the color of lightning as it fades out. Note, this is the exact color for generated lightning but should be considered a filter color for lightning that he uses bitmaps	"r g b a" (float)
strikeRadius	This value is used to specify the radius in which lightning will strike.	(0.0 , inf)
strikesPerMinute	This value is used to specify the number of strikes permitted. Note, strikes occur at random but engine guarantees that at least strikes per minute strikes will occur space every minute.	[0 , inf]
strikeWidth	This value specifies the width of the bitmap or generated lightning, at the point of its strike.	(0.0 , inf)
useFog	If such true this lightning will be affected by fog. Note, this only applies to lightning generated using bitmaps.	[false , true]

Console Method Summaries

strikeObject	strikeRandomPoint	warningFlashes

Console Methods

strikeObject(obj)

Purpose
Use the ***strikeObject*** method to cause lightning to strike the object specified object.

Syntax
obj - The name or ID of a GameBase derived object to strike with lightning.

Returns
No return value.

Notes
Not working as of this time.

See Also
strikeRandomPoint

strikeRandomPoint()

Purpose
Use the ***strikeRandomPoint*** method to generate a random lightning strike.

Returns
No return value.

See Also
strikeObject

A.2.35. LightningData

This is the datablock associated with the lightning object, it is used to specify the bitmap textures and the sounds use with the lightning object.

Fields

Field Name	Description	Sample or Range
strikeSound	Audio profile to play on strike.	AudioProfile datablock
strikeTexture[0] ... strikeTexture[7]	Texture data to use when using textured lightning.	~/path/filename.png
thunderSounds[0] ... thunderSounds[7]	Audio profiles for thunder sounds.	AudioProfile datablock

A.2.36. Marker

This marker is used for building paths used by the PathCamera and other path following objects.

Fields

Field Name	Description	Sample or Range
msToNext	Milliseconds to next marker in sequence.	(0 , inf)
seqNum	Marker position in sequence of markers on this path.	[0 , 40)
smoothingType	Path smoothing at this marker/knot. "Linear" means no smoothing, while "Spline" means to smooth.	"Linear" "Spline"
type	Type of this marker/knot. A "normal" knot will have a smooth camera translation/rotation effect. "Position Only" will do the same for translations, leaving rotation un-touched. Lastly, a "Kink" means the rotation will take effect immediately for an abrupt rotation change.	"Normal" "Position Only" "Kink"

A.2.37. MissionArea

This object is used represent the outer bounds of the mission area. This is a reactive region in that, a callback will be entered each time an object enters or leaves the mission area. Please see the callbacks quick reference for information on these callbacks.

Fields

Field Name	Description	Sample or Range
Area	Four floating point values <Xc Yc Xw YD> representing: - Center of mission area <Xc Yc> (in world). - Size of mission area <Xw Yw>	" 0.0 0.0 1024.0 512.0"
flightCeiling	This value is used to specify the elevation at which a flying vehicles power. That is, when a flying vehicle reaches this elevation it will not be able to thrust vertically any longer.	[0.0 , inf.0)
flightCeilingRange	This value is used to specify a range below the flight ceiling at which a flying vehicle's to thrust begins to taper off.	[0.0 , flightCeiling]

A.2.38. MissionMarker

This class is used to represent generic marker in the game world.

A.2.39. NetConnection

This is the parent object to GameConnection. We do not create instances of net connection, but it is important to note that this class exists and provides several features which are available through the GameConnection class.

Globals

Variable Name	Description	Sample or Range
pref::Net::PacketRateToClient	Limits the packet rate from server to client.	[1 , 32]
pref::Net::PacketRateToServer	Limits the packet rate from client to server.	[8 , 32]
pref::Net::PacketSize	Limits the size of any one packet.	[100 , 450]

Console Method Summaries

checkMaxRate	**clearPaths**	**connect**
connectLocal	**getAddress**	**getGhostID**
getGhostsActive	**getPacketLoss**	**getPing**
resolveGhostID	**resolveObjectFromGhostIndex**	**setSimulatedNetParams**

Console Methods

checkMaxRate()

Purpose
Use the ***checkMaxRate*** method to retrieve the current maximum packet rate for this connection.

Returns
Returns an integer value representing the maximum number of packets that can be transmitted by this connection per transmission period.

Notes
The period may not neccesarily be one second.
To adjust packet rates, see the preference variables above.

clearPaths()

Purpose
Use the ***clearPaths*** method to mark this connection as NOT having received any paths.

Returns
No return value.

See Also
transmitPaths

connect(remoteAddress)

Purpose
Use the ***connect*** method to request a connection to a remote server at the address ***remoteAddress***.

Syntax
remoteAddress – A string containing an address of the form: "A.B.C.D:Port", where A .. B are standard IP numbers between 0 and 255 and Port can be between 1000 and 65536.

Returns
No return value.

See Also
connectLocal, getAddress

connectLocal()

Purpose
Use the ***connectLocal*** method to connect the current client-side connection to a local NetConnection, that is to create an internal connection from this client to the internal server. This is accomplished through the use of a back door mechanism and has an extremely high bandwidth.

Returns
No return value.

See Also
connect, getAddress

getAddress()

Purpose
Use the ***getAddress*** method to get the address and port that this NetConnection is currently attached to.

Returns
Returns the address and port that this NetConnection is currently attached to, where the addres will be of the form: "A.B.C.D:Port". A .. B are standard IP numbers between 0 and 255 and Port can be between 1000 and 65536.

If the connection is local, the string "local" will be returned.

If a this NetConnection is not currently connected the method will return a NULL string.

See Also
connect, connectLocal

getGhostsActive()

Purpose
Use the ***getGhostsActive*** method to determine how many ghosts are active on a particular connection.

Returns
Returns an integer value between 0 and inf, specifying how many objects are being ghosted to a client on the other side of a specific connection.

getPacketLoss()

Purpose
Use the ***getPacketLoss*** method to determine the current packetLoss count for this connection.

Returns
Returns an integer value between 0 and inf, indicating the number of packets that have been lost to date on this net connection.

See Also
getPing

getPing()

Purpose
Use the ***getPing*** method to determine the round-trip travel time from this connection to the agent on the other end and back again.

Returns
Returns an integer value representing the total time in milliseconds it takes for a ping request to travel to the agent on the other end of a connection and back to this agent.

See Also
getPacketLoss

setSimulatedNetParams(packetLoss , delay)

Purpose
Use the ***setSimulatedNetParams*** method to force a connection to experience a certain degree of packet-loss and/or latency. This is a debug feature to allow us to see how a distributed game will behave in the face of poor connection quality.

Syntax

packetLoss	– A floating-point value between 0.0 (0%) and 1.0 (100%) dictating the percentage of packets to be artificially lost.
delay	– An integer value specifying the number of milliseconds to insert into transmission latencies.

Returns
No return value.

See Also
getPacketLoss, getPing

transmitPaths()

Purpose
Use the ***transmitPaths*** method to send Interior path information to a client on this connection.

Returns
No return value.

Notes
Only called on client connections from the server.

See Also
clearPaths

A.2.40. NetObject

Console Method Summaries

clearScopeToClient	getGhostID	scopeToClient

Console Methods

`clearScopeToClient( client )`

Purpose
Use the ***clearScopeToClient*** method to undo the effects of a previous call to *scopeToClient*.

Syntax
client – The ID of the client to stop forcing scoping this object for.

Returns
No return value.

See Also
scopeToClient

`scopeToClient( client )`

Purpose
Use the ***scopeToClient*** method to force this object to be SCOPE_ALWAYS on ***client***.

Syntax
client – The ID of the client to force this object to be SCOPE_ALWAYS for.

Returns
No return value.

Notes
When an object is SCOPE_ALWAYS it is always ghosted. Therefore, if you have an object that should always be ghosted to a client, use this method.

See Also
clearScopeToClient, setScopeAlways

setScopeAlways()

Purpose
Use the ***setScopeAlways*** method to force an object to be SCOPE_ALWAYS for all clients.

Returns
No return value.

Notes
When an object is SCOPE_ALWAYS it is always ghosted. Therefore, if you have an object that should always be ghosted to all clients, use this method.

See Also
scopeToClient

A.2.41. ParticleData (PD)

This is the datablock used to represent individual particles. This datablock is subsequently used by particle emitter data to produce particle effects. Torque provides a variety of particle effects. Particles may be animated sequences of textures, a single fixed texture, a colorized or non-colorized texture.

Fields

Field Name	Description	Sample or Range
animateTexture	Use textures beyond	[false , true]
animTexName[0] ... animTexName[49]	Textures used when animating textures. Note: animTexName[0] == textureName	~/path/filename.png
colors[0] colors[1] colors[2] colors[3]	Key-framing color controls for particles.	"1.0 0.5 0.5 1.0"
constantAcceleration	Acceleration applied to particle per second over particle lifetime.	(-inf.0 , inf.0)
dragCoefficient	Drag multiplier (total drag == particle velocity * dragCoefficient).	(-inf.0 , inf.0)
framesPerSec	Texture animation frames per second.	(-inf , inf)
gravityCoefficient	Gravity multiplier.	(-inf.0 , inf.0)
inheritedVelFactor	Velocity multiplier (inherited velocity == emitter velocity * inheritedVelFactor).	(-inf.0 , inf.0)
lifetimeMS	Time this particle lives in milliseconds. (Total lifetime == lifetimeMS +/- rand(lifetimeVarianceMS)).	(-inf , inf)
lifetimeVarianceMS	Random amount by which lifetime varies.	(-inf , inf)
sizes[0] sizes[1] sizes[2] sizes[3]	Key-framing size controls for particles.	(-inf.0 , inf.0)
spinRandomMax	Maximum degrees for randomized angular spin about billboard's normal.	(-inf.0 , inf.0)
spinRandomMin	Minimum degrees for randomized angular spin about billboard's normal.	(-inf.0 , inf.0)
spinSpeed	Degrees-per-second spin rate about billboard's normal.	(-inf.0 , inf.0)
textureName	Texture to use for non-animated particles.	~/path/filename.png

Field Name	Description	Sample or Range
times[0] times[1] times[2] times[3]	Key-framing time controls for particles.	[0.0, times[1]] [times[0], times[2]] [times[1], times[3]] [times[2], 1.0]
useInvAlpha	Invert alpha components in texture(s).	[false , true]
windCoefficient	Amount by which wind effects particle velocity.	(-inf.0 , inf.0)

A.2.42. ParticleEmitterData

Fields

Field Name	Description	Sample or Range
ejectionOffset	Each particle is given an ejection direction at which to be emitted, and ejectionOffset specifies the distance in this direction from the particle emitter a particle will be emitted at. This field's value must be non-negative, otherwise a console warning is generated and the value is forced to 0.0. The actual ejectionOffset value used is only guaranteed to be within 0.01 of the value used, and values greater than 655 are not possible.	(-inf.0 , inf.0)
ejectionPeriodMS	Used along with periodVarianceMS to determine the frequency with which to emit particles. See the periodVarianceMS field for more details. Stored with only 10 bits of precision, so values above 1023 are not possible. If value is less than 1, a console warning is generated and the value is forced to 1.	(-inf , inf)
ejectionVelocity	Used along with velocityVariance to determine the initial velocity at which a particle is emitted. See the velocityVariance field documentation for more details. This field's value must be non-negative, otherwise a console warning is generated and the value is forced to 0.0. The actual value used is only guaranteed to be within 0.01 of the value specified, and values larger than 655 are not valid.	(-inf.0 , inf.0)
lifetimeMS	Used with lifetimeVarianceMS to determine the total life-time of each emitted particle. Measured in milliseconds. This field's value must be greater than zero, otherwise a console warning is generated and the value is forced to one. To save network transmission bandwidth, the value of lifetimeMS is shifted-right 5 bits, implying a loss of precision. Thus, the lifetime value used may be up to 31 milliseconds less than the value specified. After being shifted, the value is clamped to 10-bits of precision as well; thus, values of more than about 32000 milliseconds will not behave as expected.	(-inf , inf)
lifetimeVarianceMS	Used with lifetimeMS to determine the total life-time of each emitted particle. Measured in milliseconds. Each time a particle is created, a random number between -1*lifetimeVarianceMS and lifetimeVarianceMS is generated and added to lifetimeMS to determine the particle's total lifetime. This field's value must be no greater than that of lifetimeMS, otherwise a console warning is generated and the value is clamped. The lifetimeVarianceMS value is shifted and stored with 10-bit precision in the exact manner that lifetimeMS's value is.	(-inf , inf)
orientOnVelocity	Only applicable when orientParticles is True. Specifies whether emitted particles will be checked for velocity. If True, and the particle has no velocity, it will not be rendered. If True, and the particle has a velocity, the velocity magnitude will affect the particle's rendered size. False means that the particle's velocity will not affect the rendered size.	[false , true]
orientParticles	Specifies whether emitted particles should be oriented as billboards (always face directly towards the camera), or if they should face the emission direction (see the field documentation for thetaMin, thetaMax, phiReferenceVel, and phiVariance for more information on emission direction). True means the particles should be oriented toward the emission direction, False means particles should be oriented as billboards.	[false , true]

Field Name	Description	Sample or Range
overrideAdvance	Specifies whether a newly created particle's acceleration, color, and other settings should begin being updated immediately after it is created. If false, the new particle is updated immediately. If true, the particle's time advancement system is initially deferred.	[false , true]
particles	String containing a tab-delimited list of ParticleData datablock names. When a particle is emitted from the emitter, its datablock is randomly selected from the list of valid datablocks supplied in the particles field string. The string must be non-empty and at least one token must evaluate to a valid ParticleData datablock name. String must also be less than 256 characters long. If either of these conditions is not met, a console warning will be produced and the datablock's onAdd() method will fail.	see type
periodVarianceMS	Used along with ejectionPeriodMS to determine the frequency with which to emit particles. Each time a particle is emitted, a random value between -1*periodVarianceMS and periodVarianceMS is generated and added to ejectionPeriodMS to determine the time at which to emit the next particle. So, with the default values ejectionPeriodMS = 100 and periodVarianceMS = 0, particles would be emitted exactly 10 times per second. This field's value is stored with only 10 bits of precision, so values above 1023 are not possible. Value must be less than the value of ejectionPeriodMS, otherwise a console warning is generated and the value is set to ejectionPeriodMS - 1.	(-inf , inf)
phiReferenceVel	Each time a particle is emitted, it is given an ejection direction, see the thetaMin field documentation for more information. phiReferenceVel and phiVariance determine the particle's ejection angle about the z-axis. Particles are emitted starting at 0 degrees, and the emission angle is rotated over time, according to the velocity specified in phiReferenceVel, as well as random numbers generated from phiVariance. See the phiVariance field documentation. This field's value is measured in degrees per second and may range from 0.0 to 360.0; if it lies outside this range, a console warning is generated and the value is clamped.	(-inf.0 , inf.0)
phiVariance	Used along with phiReferenceVel to determine the angle relative to the z-axis at which to eject a particle. Each time a particle is generated, a new ejection direction is determined for it. The ejection angle about the z-axis is defined by adding a random variable between 0.0 and phiVariance to the angle determined by phiRefVelocity. See the phiRefVelocity field documentation for more information. This field's value is measured in degrees and must lie in the range of 0.0 to 360.0; if it does not, a console warning is generated and the value is clamped.	(-inf.0 , inf.0)
thetaMax	Each time a particle is emitted, it is given an ejection direction. See the thetaMin field documentation for more information. thetaMax defines the maximum ejection direction angle relative to the x-axis. This field's value is measured in degrees. The value may range from 0.0 to 180.0; if it lies outside this range, a console warning is generated and the value is clamped.	(-inf.0 , inf.0)
thetaMin	Each time a particle is emitted, it is given an ejection direction. This direction is defined by the thetaMin, thetaMax, phiReferenceVel, and phiVariance fields. A random angle between thetaMin and thetaMax is generated and provides the ejection direction angle relative to the x-axis. This field's value is measured in degrees. The value may range from 0.0 to 180.0, and must be less than thetaMax. If one of these conditions is not met, a console warning is generated and the value is clamped accordingly.	(-inf.0 , inf.0)
useEmitterColors	Specifies whether the particle's datablock *colors* array field should be over-ridden by the colors array provided by the ParticleEmitter object. True implies that the particle's datablock should be over-ridden.	[false , true]
useEmitterSizes	Specifies whether the particle's datablock *sizes* array field should be over-ridden by the sizes array provided by the ParticleEmitter object. True implies that the particle's datablock should be over-ridden.	[false , true]

Field Name	Description	Sample or Range
velocityVariance	Used along with ejectionVelocity to determine the speed at which a particle will be emitted. Each particle gets a new velocity, which is calculated by adding a random number between -1*velocityVariance and velocityVariance to ejectionVelocity. This field's value must be non-negative and must not be greater than ejectionVelocity, otherwise a console warning is generated and the value is clamped appropriately. The actual value used is only guaranteed to be within 0.01 of the value specified, and values larger than 163 are not valid.	(-inf.0 , inf.0)

A.2.43. ParticleEmitterNode

Fields

Field Name	Description	Sample or Range
emitter	ParticleEmitterData datablock used by this emitter node	ParticleEmitterData datablock
velocity	Velocity at which to emit particles.	[0.0 , inf.0)

A.2.44. ParticleEmitterNodeData

Fields

Field Name	Description	Sample or Range
timeMultiple	A multiplier that will affect the behavior or the emitter node and the particles it emits.	[0.0 , inf.0)

A.2.45. Path

Fields

Field Name	Description	Sample or Range
isLooping	If this is true, the loop is closed, otherwise it is open.	[false , true]

Console Method Summaries

getPathID

Console Methods

`getPathID()`

Purpose
Use the ***getPathID*** method to get the path ID (not the object ID) of this path.

Returns
Returns an integer value representing the path index for this path as stored by the path manager.

A.2.46. PathCamera

This object is a special version of the camera that can be placed on a path. The camera can then be made to travel along this path. This feature can be used for many purposes, including cut-scenes, in-game movies, demos, etc.

Console Method Summaries

popFront	**pushBack**	**pushFront**
reset	**setPosition**	**setState**

Console Methods

popFront()

Purpose
Use the ***popFront*** method to remove a knot from the front of the camera's knot queue.

Returns
No return value.

Notes
Removes first node in path, does not affect node base or position.

See Also
pushBack, pushFront

pushBack(transform , speed , type , path)

Purpose
Use the ***pushBack*** method to add a new knot to the back of a path camera's path.

Syntax
transform - Transform vector for new knot.
speed - Speed setting for knot.
type - Knot type. ("Normal", "Position Only", "Kink")
path - Path type. ("Linear", "Spline")

Returns
No return value.

See Also
pushFront

pushFront(transform , speed , type , path)

Purpose
Use the ***pushFront*** method to add a new knot to the front of a path camera's path.

Syntax
transform - Transform vector for new knot.
speed - Speed setting for knot.
type - Knot type. ("Normal", "Position Only", "Kink")
path - Path type. ("Linear", "Spline")

Returns
No return value.

See Also
pushBack

reset([speed = 0])

Purpose
Use the ***reset*** method to move the camera back to the beginning of the path and optionally give it a new travel speed.

Syntax
speed - A positive floating-point value corresponding to the rate of travel for the camera.

Returns
No return value.

See Also
setState

setPosition(pos)

Purpose
Use the ***setPosition*** method to set the position of the camera on a path as a percentage.

Syntax
pos - A floating point value equal to the position on the current camera path. Between 0.0 and 1.0.

Returns
No return value.

See Also
setState, setTarget

setState(state)

Purpose
Use the ***setState*** method to set the camera's current movement state.

Syntax
state – A string containing either:
```
        forward  – moving from front of path to back of path.
        backward – moving from back of path to front of path.
        stop     – not moving.
```

Returns
No return value.

See Also
reset, setPosition

setTarget(pos)

Purpose
Use the ***setTarget*** method to set the next position target for this path camera. A path camera will travel along the path until it reaches the current target. This target is set as a percentage of the entire path.

Syntax
pos – A floating point value represent the next target on this path camera's path. Between 0.0 and 1.0.

Returns
No return value.

Notes
Setting a target position behind the current path camera position will not make the camera travel in reverse unless the moving state is backward.

See Also
setPosition, setState

A.2.47. PhysicalZone

Fields

Field Name	Description	Sample or Range
appliedForce	Three-element floating point value representing forces in three axes to apply to objects entering pzone.	"x y z" Each value in range: [-40000 , 40000]
gravityMod	Gravity in pzone. Multiplies against standard gravity.	(-inf.0 , inf.0)
polyhedron	Floating point values describing outer bounds of pzone.	-
velocityMod	Multiply velocity of objects entering zone by this value every tick.	[-40000 , 40000]

Console Method Summaries

activate	deactivate

Console Methods

activate()

Purpose
Use the ***activate*** method to enable this physicalZone. Once enabled, all of the physicalZone attributes will be in effect.

Returns
No return value.

See Also
deactivate

deactivate()

Purpose
Use the ***deactivate*** method to disable this physicalZone. Once disabled, none of the physicalZone attributes will be in effect.

Returns
No return value.

See Also
activate

A.2.48. Player

Console Method Summaries

checkDismountPoint	**clearControlObject**	**getControlObject**
getDamageLocation	**getState**	**setActionThread**

Console Methods

checkDismountPoint(oldPos , pos)

Purpose
Use the ***checkDismountPoint*** method to see if dismounting a player from one position to another will cause a collision, or a cohabitation of space.

Syntax
oldPos – Original position of player.
pos – Intended position of player.

Returns
Returns true if the new position does not result in any collisions, false otherwise.

Notes
This can be used for a variety of things besides dismounting checks. Basically, any time you want to check if a player can be moved into a new position, you may use this.

clearControlObject()

Purpose
Use the ***clearControlObject*** method to undo the results of a call to *player.setControlObject*, re-establishing this player as the control object.

Returns
No return value.

Notes
It is possible to temporarily make another player receive move commands that would normally be sent to this player, while leaving this player in place. Unless moved, the camera will stay mounted to this player, while the other player receives movement inputs.

See Also
getControllingObject, setControlObject

Description

getControllingObject()

Purpose
Use the ***getControllingObject*** method to determine if this player object is receiving input commands from another player.

Returns
Returns 0 if this object is not being controlled by another player object, or it will return a non-zero positive integer specifying the ID of the player object that is controlling this player.

Notes
It is possible to temporarily make another player receive move commands that would normally be sent to this player, while leaving this player in place. Unless moved, the camera will stay mounted to this player, while the other player receives movement inputs.

See Also
clearControlObject, setControlObject

Description

Returns float

getDamageLocation(damagePosition)

Purpose
Use the ***getDamageLocation*** method to get the named damage location and modifier for a given ***damagePosition***. The player object can differentiate 'hit' locations based on a pre-defined set of datablock settings. You may modify these settings, to vary the percentage of area given over to any body region.

Syntax
damagePosition – A position for which to retrieve a body region on this player.

Returns
Returns a string containing two words (space separated strings), where the first is a location and the second is a modifier:

Posible locations
- legs
- torso
- head

Head modifiers
- front_left
- front_right
- back_left
- back_right

```
   Legs/Torso modifiers
   - middle_back
   - right_back
   - left_middle
   - middle_middle
   - right_middle
   - left_front
   - middle_front
   - right_front
```

Notes
Positions do not need to be on or even near the player. The engine will calculate which body location and modifier best fit any position relative to the player.

getState()

Purpose
Use the ***getState*** method to get the current state of this player.

Returns
Returns a string specifying the current state of the player:

```
   Possible states
   Dead    - Damage >= disabledLevel
   Mounted - Mounted to another object.
   Move    - Alive and able to move.
   Recover - Recovering from a long fall or hard impact (unable to move).
```

Notes
The most important fact to note is that your player is considered dead at the disabledLevel, not the destroyedLevel.

setActionThread(string sequenceName , hold , fps)

Purpose
Use the ***setActionThread*** method to play a specific animation. This is similar to the *playThread* method that comes with all ShapeBase objects, but it is designed for full body animations and comes with a few different options. Optionally, a player may be told to play a thread and then to hold at the last frame of that animation until another over-rides it.

Syntax
sequenceName - A string containing the name of the animation to play.

Returns
No return value.

setControlObject(obj)

Purpose
Use the ***setControlObject*** method to have this player object 'control' another player object. Doing so will re-direct movement inputs from this player object to the controlled player object. This can be used when you want one player to act as a mount for another player.

Syntax
obj – The ID of the surrogate player that should now receive movement inputs previously destined for this player.

Returns
Returns true on success, and false on failure.

Notes
It is possible to temporarily make another player receive move commands that would normally be sent to this player, while leaving this player in place. Unless moved, the camera will stay mounted to this player, while the other player receives movement inputs.

See Also
clearControlObject, getControllingObject

A.2.49. PlayerData

Fields

Field Name	Description	Sample or Range
Rendering		
renderFirstPerson	If true, render the player mesh while in 1st POV.	[false , true]
Pickups		
pickupRadius	Can be used to increase the size of the player box for item collisions ONLY.	[0.0 , inf.0)
Looking		
maxFreelookAngle	Maximum free-look angle in radians.	[-3.14159 , 3.14159]
maxLookAngle	Maximum look angle in radians.	[-3.14159 , 3.14159]
minLookAngle	Minimum look angle in radians.	[-3.14159 , 3.14159]
Time Scaling		
maxTimeScale	Limits animation scaling. Animations will scale up to this amount when matching ground velocity.	[0.0 , inf.0)
Step Height		--
maxStepHeight	Maximum height player will be able to step up. Heights larger than this will stop a walking player.	[0.0 , inf.0)
Forces and Factors		
horizMaxSpeed	Maximum horizontal velocity on ground, in air, or in water.	--
horizResistFactor	Delta factor used to determine how much of 'horizResistspeed' is removed from current velocity.	--
horizResistSpeed	Velocity at which horizontal resistance kicks in.	--
jumpDelay	Forced delay between jumps (in ticks).	--
jumpEnergyDrain	Drain this many energy points for every jump.	--
jumpForce	Force applied to player on jump. Should be less than 40000 * mass.	--

Field Name	Description	Sample or Range
jumpSurfaceAngle	Cannot jump if surface angle equal to or greater to this many degrees.	--
maxJumpSpeed	Cannot jump if running faster than this.	--
minJumpEnergy	Cannot jump if energy lower than this.	--
minRunEnergy	Cannot run if energy lower than this.	--
recoverDelay	Number of ticks that player stays in 'recovery mode' after hard impact.	[0 , inf]
recoverRunForceScale	Run force is multiplied by this amount during 'recovery'.	[0.0 , 1.0]
runEnergyDrain	Drain this much energy per tick while running.	--
runForce	Accelerate player by this much per tick as a result of a move (command). Should be less than 40000 * mass.	--
runSurfaceAngle	Cannot accelerate if surface angle equal to or greater to this many degrees.	--
upMaxSpeed	Maximum velocity allowed in the positive Z direction.	--
upResistFactor	Delta factor used to determine how much of 'upResistSpeed' is removed from current velocity.	--
upResistSpeed	Velocity at which vertical resistance kicks in.	--
Velocity		
maxBackwardSpeed	Maximum backward velocity in m/s.	--
maxForwardSpeed	Maximum forward velocity in m/s.	--
maxSideSpeed	Maximum sideway velocity in m/s.	--
maxUnderwaterBackwardSpeed	Maximum underwater backward velocity in m/s.	--
maxUnderwaterForwardSpeed	Maximum underwater forward velocity in m/s.	--
maxUnderwaterSideSpeed	Maximum underwater sideway velocity in m/s.	--
Impacts		
groundImpactMinSpeed	The velocity at which a collision with the ground is registered as an impact.	[0.0 , inf.0)
groundImpactShakeAmp	Camera shake amplitude when a ground impact is registered.	"1.0 1.0 10.0"
groundImpactShakeDuration	Camera shake duration when a ground impact is registered.	[0.0 , inf.0)
groundImpactShakeFalloff	a multiplier determining at what rate the shaking from an impact is reduced over time.	[0.0 , inf.0)
groundImpactShakeFreq	Camera shake frequency when a ground impact is registered.	"4.0 4.0 4.0"
minImpactSpeed	The velocity at which a collision with the any object is registered as an impact.	[0.0 , inf.0)
Hit Boxes		
boundingBox	A vector containing the radius values for each dimension of the unscaled player's bounding box. (Will be multiplied by scale.)	"1.6 1.6 2.3"
boxHeadBackPercentage	In conjunction with boxHeadBackPercentage, used to determine whether the front, middle, or back side of the player's torso or legs was hit. Note: this functionality is difficult to properly change; it is recommended that default values be used.	0
boxHeadFrontPercentage	In conjunction with boxHeadBackPercentage, used to determine whether the front, middle, or back side of the player's torso or legs was hit. Note: this functionality is difficult to properly change; it is recommended that default values be used.	1
boxHeadLeftPercentage	In conjunction with boxHeadRightPercentage, used to determine whether the left, middle, or right side of the player's torso or legs was hit. Note: this functionality is difficult to properly change; it is recommended that default values be used	0

Field Name	Description	Sample or Range
boxHeadPercentage	The distance from the bottom of the player's bounding box, to the beginning of the area considered to bound the player's head. Measured as a raw percentage.	[0.0 , 1.0]
boxHeadRightPercentage	In conjunction with boxHeadRightPercentage, used to determine whether the left, middle, or right side of the player's torso or legs was hit. Note: this functionality is difficult to properly change; it is recommended that default values be used	1
boxTorsoPercentage	Like boxHeadPercentage, but specifies where the torso area begins.	[0.0 , 1.0]
Footprints and -puffs		
decalData	DecalData datablock used for footprints.	
decalOffset	The offset from the player's center by which the decal should be displayed; should correlate with the distance from player's center, to the player's right foot center.	
dustEmitter	--	Not Used
footPuffEmitter	The ParticleEmitterData datablock will be used to emit particles at footstep locations.	
footPuffNumParts	Number of dust particles to be generated for foot puffs.	[0 , inf)
footPuffRadius	The radius of dust spread from foot steps.	[0.0 , inf.0)
Sounds and Modifiers		
exitingWater	The AudioProfile used to produce sounds when emerging from water.	--
exitSplashSoundVelocity	The minimum velocity at which the exit splash sound will be played when emerging from water.	--
FootBubblesSound	The AudioProfile used to produce sounds for bubbles produced by footsteps.	--
FootHardSound	The AudioProfile used to produce sounds for hard footsteps.	--
FootMetalSound	The AudioProfile used to produce sounds for soft footsteps.	--
FootShallowSound	The AudioProfile used to produce sounds for shallow water footsteps.	--
FootSoftSound	The AudioProfile used to produce sounds for soft footsteps.	--
FootUnderWater	AudioProfileThe AudioProfile used to produce sounds for underwater footsteps.	--
Footwading	The AudioProfile used to produce sounds for wading footsteps.	--
hardSplashSoundVelocity	The minimum velocity at which the hard splash sound will be played when entering or leaving water.	--
impactHardSound	The AudioProfile used to produce sounds for hard impacts.	--
impactMetal	The AudioProfile used to produce sounds for metal impacts.	--
impactSnowSound	The AudioProfile used to produce sounds for snow impacts.	--
impactSoftSound	The AudioProfile used to produce sounds for soft impacts.	--
impactWaterEasy	The AudioProfile used to produce sounds for soft impacts against water.	--
impactWaterHard	The minimum velocity at which the hard water impact sound will be played when entering or leaving water.	--
impactWaterMedium	The AudioProfile used to produce sounds for medium impacts against water.	--
mediumSplashSoundVelocity	The minimum velocity at which the medium water impact sound will be played when entering or leaving water.	--
movingBubblesSound	The AudioProfile used to produce sounds for underwater bubble while the player is moving.	--
waterBreathSound	The AudioProfile used to produce sounds for underwater breathing.	--
Splashes and Bubbles		
bubbleEmitTime	The length of time to continue emitting bubbles.	--

Field Name	Description	Sample or Range
footstepSplashHeight	The maximum height at which to play the shallow footstep effect.	--
splash	The SplashData datablock used to emit splashes.	--
splashAngle	The mimimum verticle angle at which a player must be traveling in order to generate a splash effect.	--
splashEmitter[0] splashEmitter[1]	Array of pointers to ParticleEmitterData datablocks which will be used to generate splash effect particles.	--
splashFreqMod	The simulated frequency modulation of a splash generated by this player. Multiplied along with player speed and time elapsed when determining splash emission rate.	--
splashVelEpsilon	The threshold speed at which we consider the player's movement to have stopped when updating splash effects.	--
splashVelocity	The minimum velocity a player needs in order to generate a splash effect.	--

A.2.50. Precipitation

Fields

Field Name	Description	Sample or Range
boxHeight	Height of box (around camera) where precipitation occurs.	(0.0 , inf)
boxWidth	Width of box (around camera) where precipitation occurs.	(0.0 , inf)
doCollision	Allow precipitation to collide with objects vs. fall through.	[false , true]
maxMass	Maximum mass per drop.	[minMass , inf)
maxSpeed	Maximum speed per drop.	[minSpeed , inf)
maxTurbulence	Max turbulence to apply per drop.	[0.0 , inf)
minMass	Minimum mass per drop.	[0.0 , maxMass]
minSpeed	Minimum speed per drop.	[0.0 , maxSpeed]
numDrops	Approximate number of drops allowed to exist at any one time.	(0 , inf)
rotateWithCamVel	Drops rotate to face camera.	[false , true]
turbulenceSpeed	Velocity (of drops) at which turbulence kicks in.	[0.0 , inf)
useTurbulence	Enable turbulence effect.	[false , true]

Console Method Summaries

modifyStorm	**setPercentage**

Console Methods

modifyStorm(*percentage* , *time*)

Purpose
Use the ***modifyStorm*** method to adjust the ***percentage*** of raindrops falling over ***time*** period.

Syntax
percentage – Percent of raindrops to render. [0.0 , 1.0]
time – Period of transition in seconds.(0.0 , inf)

Returns
No return value.

See Also
setPercentage

setPercentage(*percentage*)

Purpose
Use the ***setPercentage*** method to immediately adjust the ***percentage*** of raindrops falling.

Syntax
percentage – Percent of raindrops to render. [0.0 , 1.0]

Returns
No return value.

See Also
modifyStorm

A.2.51. PrecipitationData

Fields

Field Name	Description	Sample or Range
dropSize	Render size for drops.	(0.0 , inf)
dropTexture	Texture file (4 x 4 bitmap array) to use for drops.	texture Path
soundProfile	Looping 2D audio profile to play with precipitation.	Audio Profile Name
splashMS	Life of splashes in milliseconds.	(0 , inf)
splashSize	Size of splashes.	(0.0 , inf)
splashTexture	Texture to use for splashes (4 x 4 bitmap array).	texture path
useTrueBillboards	Drops behave like true (non axis-aligned) billboards.	[false , true]

A.2.53. Projectile

Fields

Field Name	Description	Sample or Range
initialPosition	Position that projectile starts at.	"1.0 2.0 3.0"
initialVelocity	Starting velocity of projectile.	"1.0 2.0 3.0"
sourceObject	description	(-inf , inf)
sourceSlot	description	(-inf , inf)

A.2.53. ProjectileData

Fields

Field Name	Description	Sample or Range
armingDelay	The number of ticks that must pass after the Projectile is created before it can explode. Valid range is 0 to Projectile::MaxLivingTicks (4095 by default).	(-inf , inf)
bounceElasticity	Used to simulate the Projectile's bounce elasticity, if it collides with something but does not explode. The bounce elasticity scales the velocity from a bounce, after friction is taken into account.	[0.0 , 0.999]
bounceFriction	On bounce, reduce projectile velocity by this factor and a multiple of the tangent to impact.	(-inf.0 , inf.0)
decals[0] ... decals[5]	Array of pointers to DecalData datablocks. A non-NULL decal object will be randomly chosen from the array when the Projectile collides with the terrain or an interior.	DecalData datablock
explosion	ExplosionData datablock used when the Projectile object blows up out of water.	see type
fadeDelay	The number of ticks that must pass after the Projectile is created before it will begin becoming transparent. Projectile's opacity will follow a linear degression starting fadeDelay number of ticks after it is creating and ending at lifetime number of ticks. Valid range is 0 to Projectile::MaxLivingTicks (4095 by default).	(-inf , inf)
gravityMod	If isBallistic is true, scales the effect of gravity on the Projectile. Valid range is 0.0 to 1.0.	(-inf.0 , inf.0)
hasLight	Specifies whether the Projectile object sheds light when not in water. If so, a point light object is used with the radius and color specified in the lightRadius and lightColor fields.	[false , true]
hasWaterLight	Specifies whether the Projectile object sheds light when in water. If so, a point light object is used with the radius and color specified in the lightRadius and waterLightColor fields.	[false , true]
isBallistic	Specifies whether the Projectile will be affected by gravity, and whether it can bounce before exploding.	[false , true]
lifetime	The number of ticks the Projectile should survive for. Also used along with fadeDelay to determine the transparency of the Projectile object at a given time. See the fadeDelay field documentation for more information. The valid range for lifetime values is 0 to Projectile::MaxLivingTicks (4095 by default).	(-inf , inf)
lightColor	The projectile's point light color for use when not in water.	"1.0 0.5 0.5"
lightRadius	The projectile's point light radius. Valid range is 1.0 to 20.0	(-inf.0 , inf.0)

Field Name	Description	Sample or Range
muzzleVelocity	*Note: this field currently has no tangible effect in the engine simulation itself, but it is useful in script, and its valid range is checked by the engine.* Valid range is from 0.0 to 10,000.0	(-inf.0 , inf.0)
particleEmitter	ParticleEmitter datablock used to generate particles for the projectile when it is out of water, and when entering or leaving water.	see type
particleWaterEmitter	ParticleEmitter datablock used to generate particles for the projectile when it is under water, and when it is entering or leaving water.	see type
projectileShapeName	The name of the shape file for the Projectile object. Must adhere to the semantics associated with the Filename datatype as defined in the engine.	~/path/filename.dts
sound	AudioProfile used to generate the Projectile object's sound.	see type
Splash	SplashData datablock used in the generation of splash effects when the Projectile is entering or leaving water.	see type
velInheritFactor	*Note: this field currently has no tangible effect in the engine simulation itself, but it is useful in script, and its valid range is checked by the engine.* Valid range is from 0.0 to 1.0.	(-inf.0 , inf.0)
waterExplosion	ExplosionData datablock used when the Projectile object blows up in the water.	see type
waterLightColor	The projectile's point light color for use when in water.	"1.0 0.5 0.5"

A.2.54. SceneObject

Console Method Summaries

getForwardVector	**getObjectBox**	**getPosition**
getScale	**getTransform**	**getWorldBox**
getWorldBoxCenter	**setScale**	**setTransform**

Console Methods

getForwardVector()

Purpose
Use the ***getForwardVector*** method to get the three-element floating-point vector representing the direction the object is facing.

Returns
Returns a three-element floating-point vector representing the direction the object is facing.

Notes
Forward for all SceneObjects is positive-Y. So, this vector represents the direction the SceneObject's positive-Y axis is pointing.

getObjectBox()

Purpose
Use the ***getObjectBox*** method to get the six-element floating-point vector containing two three-space points representing the unscaled and unrotated bounding box for this object.

Returns
Returns a six-element floating-point vector containing two three-space points representing the bounds of this box:

"minX minY minZ maxX maxY maxZ"

Notes
This box is unscaled and represents the bounding box of the exported, pre-scaled object.

See Also
getWorldBox

getPosition()

Purpose
Use the ***getPosition*** method to get the current position of this object.

Returns
A three-element vector containing the XYZ world position of this SceneObject.

See Also
getTransform, setTransform

getScale()

Purpose
Use the ***getScale*** method to get the scale of this SceneObject.

Returns
Returns a three-element vector containing the XYZ scale of this SceneObject.

See Also
setScale

getTransform()

Purpose
Use the ***getTransform*** method to get the transform matrix for this SceneObject.

Returns
Returns a seven-element matrix/vector containing the following information:

" PosX PosY PoxZ RotX RotY RotZ theta "

, where theta is a rotation about the axis formed by " RotX RotY RotZ ".

Notes
Use the **getWord()**, **getWords()**, and **setWord()** string functions for parsing the transform vector.

See Also
setTransform

getWorldBox()

Purpose
Use the ***getWorldBox*** method to get the six-element floating-point vector containing two three-space points representing the scaled and unrotated bounding box for this object.

Returns
Returns a six-element floating-point vector containing two three-space points representing the bounds of this box:

"minX minY minZ maxX maxY maxZ"

Notes
This box is scaled such that it will contain all points on the current object, regardless of scaling and rotation. Thus, as an irregularly shaped object rotates, its world box will change.

See Also
getObjectBox

getWorldBoxCenter()

Purpose
Use the ***getWorldBoxCenter*** method to get the centroid of this objects world box.

Returns
Returns a three-element position vector representing the centroid of this objects's world box.

See Also
getWorldBox

setScale(*scale*)

Purpose
Use the ***setScale*** method to set the XYZ scaling factor for this object.

Syntax
scale – An XYZ vector containing the new scaling factor for this object.

Returns
No return value.

See Also
getScale

setTransform(*transform*)

Purpose
Use the ***setTransform*** method to apply a new transform to this object.

Syntax
transform – A seven-element matrix/vector containing the following information:

" PosX PosY PoxZ RotX RotY RotZ theta "

, where theta is a rotation about the axis formed by " RotX RotY RotZ ".

Returns
No return value.

Notes
This will both translate and rotate the object.

See Also
getTransform

A.2.55. ScriptGroup

Fields

Field Name	Description	Sample or Range
class	A new namespace to place after the object's name and before ScriptObject in the namespace chain.	"Rectangle"
superClass	A new namespace to place after class and before ScriptObject in the namespace chain.	"Polyhedron"

A.2.56. ScriptObject

Fields

Field Name	Description	Sample or Range
class	A new namespace to place after the object's name and before ScriptObject in the namespace chain.	"Rectangle"
superClass	A new namespace to place after class and before ScriptObject in the namespace chain.	"Polyhedron"

A.2.57. ShapeBase

Globals

Variable Name	Description	Sample or Range
SB::DFDec	Damage flash reduced by this amount per tick.	(0.0, 1.0]
SB::WODec	Whiteout reduced by this amount per tick.	(0.0, 1.0]
pref::environmentMaps	Enables environmental mapping for all shapes.	[false , true]

Console Functions

setShadowDetailLevel()

setShadowDetailLevel(level)

Purpose
Use the ***setShadowDetailLevel*** function to modify the detail level of dynamically cast shadows.

Syntax
level – A floating-point value between 0.0 and 1.0, where 0.0 is low-quality and 1.0 is high quality.

Returns
No return value.

Console Method Summaries

applyDamage	applyImpulse	applyRepair
canCloak	getAIRepairPoint	getCameraFov
getControllingClient	getControllingObject	getDamageFlash
getDamageLevel	getDamagePercent	getDamageState
getEnergyLevel	getEnergyPercent	getEyePoint
getEyeTransform	getEyeVector	getImageAmmo
getImageLoaded	getImageSkinTag	getImageState
getImageTrigger	getMountedImage	getMountedObject
getMountedObjectCount	getMountedObjectNode	getMountNodeObject
getMountSlot	getMuzzlePoint	getMuzzleVector
getObjectMount	getPendingImage	getRechargeRate
getRepairRate	getShapeName	getSkinName
getSlotTransform	getVelocity	getWhiteOut
isCloaked	isDestroyed	isDisabled
isEnabled	isHidden	isImageFiring
isImageMounted	isMounted	mountImage
mountObject	pauseThread	playAudio
playThread	setCameraFov	setCloaked
setDamageFlash	setDamageLevel	setDamageState
setDamageVector	setEnergyLevel	setHidden
setImageAmmo	setImageLoaded	setImageTrigger
setInvincibleMode	setRechargeRate	setRepairRate
setShapeName	setSkinName	setThreadDir
setVelocity	setWhiteOut	startFade
stopAudio	stopThread	unmount

Console Methods

Cloaking

canCloak	isCloaked	setCloaked

canCloak()

Purpose
Use the ***canCloak*** method to determine if this shape is able to cloak.

Returns
Returns true if this shape is allowed to cloak, false otherwise.

See Also
isCloaked, setCloaked

isCloaked()

Purpose
Use the ***isCloaked*** method to determine if this shape is currently cloaked.

Returns
Returns true if the shape is currently cloaked.

See Also
canCloak, setCloaked

setCloaked(isCloaked)

Purpose
Use the ***setCloaked*** method to cloak or uncloak the current shape.

Syntax
isCloaked – A boolean value. If set to true, this shape will be cloaked, otherwise it will be uncloaked.

Returns
No return value.

See Also
canCloak, isCloaked

Hiding and Fading

isHidden	setHidden	startFade

isHidden()

Purpose
Use the ***isHidden*** method to see if this shape is currently hidden.

Returns
Returns true if the object is hidden, false otherwise.

See Also
setHidden

Returns ID of this object's datablock.

setHidden(*isHidden*)

Purpose
Use the ***setHidden*** method to hide or unhide this shape.

Syntax
isHidden – A boolean value. If set to true, this shape will be hidden, otherwise it will be un-hidden.

Returns
No return value.

Notes
When an object is hidden it is temporarily removed from the scene and therefore will not render or be collided with.

See Also
isHidden

startFade(***time*** **,** ***delay*** **,** ***fadeOut*** **)**

Purpose
Use the ***startFade*** method to fade this shape in and out of view without removing it from the scene.

Syntax
time - specifies the timeit takes (in milliseconds) for the fade to complete.
delay - specifes the time to wait (in milliseconds) before starting to fade.
fadeOut - If true, shape fades out, else shape fades in.

Returns
No return value.

Notes
Items have the ability to light their surroundings. When an Item with an active light is fading out, the light it emits is correspondingly reduced until it goes out. Likewise, when the item fades in, the light is turned-up till it reaches it's normal brightness.
A faded out object is still in the scene and can still be collided with, so if you want to disable collisions for this shape after it fades out use *setHidden* to temporarily remove this shape from the scene.

See Also
setHidden

Re-Skinning

getSkinName	setSkinName

getSkinName()

Purpose
Use the **getSkinName** method to determine which skin this shape is currently using.

Returns
Returns a string containing the name of the skin this shape is using, or if this shape is not set up to use multiple skins, it returns the NULL string.

Notes
Shapes must be skinned with specially named set of textures in order to enable re-skinning. Please see GPGT volume 1 for more details.

See Also
setSkinName

setSkinName(*skinName*)

Purpose
Use the ***setSkinName*** method to change the current skin for this shape.

Syntax
skinName – A string containing a valid skin name, as set up in the shapes set of skins. If this skin name does not match any of the textures used by this object, the object will not change skins.

Returns
No return value.

Notes
Shapes must be skinned with specially named set of textures in order to enable re-skinning. Please see GPGT volume 1 for more details.

See Also
getSkinName

Damage

applyDamage	applyRepair	getAIRepairPoint	getDamageLevel
getDamagePercent	getDamageState	isEnabled	isDisabled
isDestroyed	setDamageLevel	setDamageState	setDamageVector
setInvincibleMode	setRepairRate		

applyDamage(*damage*)

Purpose
Use the ***applyDamage*** method to apply the specified amount of ***damage*** to this shape.

Syntax
damage – A floating-point value specifying a positive amount of damage to apply to this shape.

Returns
No return value.

See Also
applyRepair, getDamageLevel

applyRepair(*repair*)

Purpose
Use the ***applyRepair*** method to apply ***repair*** amount or repair to shape.

Syntax
repair – A positive number of repair points to apply to this shape.

Returns
No return value.

Notes
Before this will work, the self-repair must at least temporarily be disabled, by calling *setRepairRate* with an argument of 0.

See Also
applyDamage, getDamageLevel, setRepairRate

getAIRepairPoint()

Purpose
Use the ***getAIRepairPoint*** method to get the the position of a specially named mesh node: "AIRepairNode". It is used as a helper when deciding where an AI should 'stand' to repair this shape.

Returns
Returns a three-element floating-point vector containing the position of a specially named mesh node: "AIRepairNode". It is used as a helper when deciding where an AI should 'stand' to repair this shape.

Notes
If this node is not present in the shape, the position of the shape's centroid will be returned instead.

getDamageLevel()

Purpose
Use the ***getDamageLevel*** method to determine the current level of damage this shape has sustained.

Returns
Returns a floating point value between 0.0 and maxDamage (as specified in the shape's datablock). 0.0 is equal to 'no damage'.

See Also
applyDamage, applyRepair, getDamagePercent, getDamageState

getDamagePercent()

Purpose
Use the ***getDamagePercent*** method to get a relative percentage of damage for this shape, where the equation for this percent is currentDamage / maxDamage.

Returns
Returns a floating point value between 0.0 (0%) and 1.0 (100%).

See Also
applyDamage, applyRepair, getDamageLevel, getDamageState

getDamageState()

Purpose
Use the ***getDamageState*** method to determine this shape's current damage state.

Returns
Returns a string specifying the current damage state for this shape:

Enabled – Damage < Disabled Level
Disabled – Disabled Level <= Damage < Destroyed Level
Destroyed – *DestroyedLevel <= Damage*

See Also
applyDamage, applyRepair, getDamageLevel, getDamagePercent

isEnabled()

Purpose
Use the ***isEnabled*** method to determine if this shape's damage is less than its disabled level.

Returns
Returns true as long as this shape's damage level is less than its disabled level.

See Also
getDamageLevel, getDamagePercent, getDamageState, isDisabled

isDisabled()

Purpose
Use the ***isDisabled*** method to determine if this shape's damage is greater than or equal to its disabled level.

Returns
Returns true as long as this shape's damage level is greater than or equal to its disabled level.

See Also
getDamageLevel, getDamagePercent, getDamageState, isEnabled

isDestroyed()

Purpose
Use the ***isDestroyed*** method to determine if this shape's damage is greater than or equal to its destroyed level.

Returns
Returns true as long as this shape's damage level is greater than or equal to its destroyed level.

See Also
getDamageLevel, getDamagePercent, getDamageState, isEnabled, isDisabled

setDamageLevel(*level*)

Purpose
Use the ***setDamageLevel*** method to set the shape's damage to a new ***level***.

Syntax
level – A floating-point value between 0.0 and inf.0, representing this shape's new damage level.

Returns
No return value.

Notes
If ***level*** is greater than this shape's maxDamage, the damage level will be capped at maxDamage. Setting this value will also update the shape's damage state.

See Also
getDamageLevel, getDamagePercent, getDamageState

setDamageState(*state*)

Purpose
Use the ***setDamageState*** method to change this shape's damage ***state***.

Syntax
state – A string containing one of the allowed damage states: enabled, disabled, or destroyed.

Returns
Returns true on success, false otherwise.

Notes
Setting this value will <u>NOT</u> affect the shape's damage level, furthermore the next time this shape's damage level is updated, the shape's damage state will change to match it.

See Also
setDamageLevel

setDamageVector(*damageOrigin*)

Purpose
Use the ***setDamageVector*** method to establish the direction from which to be applied damage will be coming. This will subsequently be used to correctly setup any explosion or debris scatter that occcurs as a result of the damage.

Syntax
damageOrigin – An XYZ vector specifying the direction from which some to be applied damage originated.

Returns
No return value.

Notes
Generally, this should be done before applying damage, but it can be done afterward. However results may vary.

See Also
applyDamage

setInvincibleMode(*time* , *speed*)

Purpose
Use the ***setInvincibleMode*** method to temporarily make this shape invincible. i.e. Not able to take damage. While the player is invincible, the screen will flicker blue with a varying rate and a varying intensity.

Syntax
time – A floating-point value specifying the time in seconds for this shape to remain invincible.
speed – A floating-point value between 0.0 and 1.0 controlling the rate at which the blue flickering effect occurs.

Returns
No return value.

Notes
The flickering effect is used to indicated to a player that his (or her) avatar is invincible. Furthermore, this flicker rate will change and the flicker will become increasingly translucent as the time elapses. If ***speed*** set to 1.0, the flickering is a bit obscene. Generally, lower values are nicer.

setRepairRate(*rate*)

Purpose
Use the ***setRepairRate*** method to allow this shape to auto-repair.

Syntax
rate – A floating-point value specifying the number of points to heal per tick, where a tick is by default 1/32 of a second.

Returns
No return value.

Notes
If you wish to manually repair a shape using *applyRepair*, you will temporarily have to disable auto-repair by calling this method with an argument of 0.

See Also
applyRepair

Damage Flashes & Whiteouts

getDamageFlash	getWhiteOut	setDamageFlash	setWhiteOut

getDamageFlash()

Purpose
Use the ***getDamageFlash*** method to get the current level of damage flash occuring, if any.

Returns
Returns a floating-point value between 0.0 and 1.0 specifying the current level of damage flash occuring.

Notes
A damage flash is a translucent red overlay that can be used to inform the player that their avatar has taken damage. When a damage flash is applied, it will fade slowly over time. This method merely lets us know how far along that fade has progressed.

See Also
setDamageFlash

getWhiteOut( )

Purpose
Use the ***getWhiteOut*** method to get the current level of whiteout occurring, if any.

Returns
Returns a floating-point value between 0.0 and 1.0 specifying the current level of whiteout occurring.

Notes
A whiteout is a translucent white overlay that can be used to temporarily blind the player. Usually this is done as a response to the player viewing a bright light of some sort, but it must be done manually if we want this effect.

See Also
setWhiteOut

setDamageFlash(*level*)

Purpose
Use the ***setDamageFlash*** method to set the current damage flash ***level***. When this level is > 0.0 the screen is rendered with an an additive red overlay, which is limited to 76% regardless of ***level***. The value of ***level*** is auto-decremented over a few seconds.

Syntax
level – A floating-point value specifying the level of damage flash to apply. This value can be between 0.0 (0%) and 1.0 (100%).

Returns
No return value.

Notes
A damage flash is a translucent red overlay that can be used to inform the player that their avatar has taken damage.

See Also
getDamageFlash

setWhiteOut(*level*)

Purpose
Use the ***setWhiteOut*** method to set the current whiteout ***level***. When this level is > 0.0 the screen is rendered with an additive white overlay. At 1.0, rendering is saturated and the screen is entirely white. i.e. it is whited out. The value of ***level*** is auto-decremented over a few seconds.

Syntax
level – A floating-point value specifying the level of whiteout to apply. This value can be between 0.0 (0%) and 1.0 (100%).

Returns
No return value.

Notes
A whiteout is a translucent white overlay that can be used to temporarily blind the player.

See Also
getWhiteOut

Energy

getRechargeRate	getEnergyLevel	getEnergyPercent	setRechargeRate
setEnergyLevel			

getRechargeRate()

Purpose
Use the ***getRechargeRate*** method to get the current rechargeRate setting for this shape.

Returns
Returns a floating-point value between 0.0 and inf.0, equivalent to the number of energy points this shape will auto-recharge every tick, where a tick is by default 1/32 of a second.

See Also
setRechargeRate

getEnergyLevel()

Purpose
Use the ***getEnergyLevel*** method to return the current energy level of this shape.

Returns
Returns a floating-point value between 0.0 and maxEnergy.

See Also
getEnergyPercent, setEnergyPercent

getEnergyPercent()

Purpose
Use the ***getEnergyPercent*** method to return the current energy level as a percentage.

Returns
Returns a floating-point value between 0.0 (0%) and 1.0 (100%), where this percentage is calculated as: currentEnergy / maxEnergy.

See Also
getEnergyPercent, setEnergyLevel

setRechargeRate(*rate*)

Purpose
Use the ***setRechargeRate*** method to enable auto-recharging at **rate** points per tick.

Syntax
rate – A floating-point value between 0.0 and inf.0 equivalent to the number of energy points to recharge this shape by each tick, where a tick is by default 1/32 of a second.

Returns
No return value.

Notes
Having auto-rechage enabled does not prevent manual application of recharge values, as is the case with auto-repair and manual repairs.

See Also
getRechargeRate

setEnergyLevel(*level*)

Purpose
Use the ***setEnergyLevel*** method to set the shape's current energy level to a value between 0.0 and maxEnergy.

Syntax
level – A floating-point value between 0.0 and maxEnergy.

Returns
No return value.

Notes
If ***level*** is greater than maxEnergy, maxEnergy is used instead.

See Also
getEnergyLevel

Eye Transform and Vectors

getEyePoint	getEyeTransform	getEyeVector

getEyePoint()

Purpose
Use the ***getEyePoint*** method to get the positions of this shape's 'eye' node.

Returns
If this shape has a node in it's mesh named 'eye', the position of that node will be returned as a three-element floating-point vector. If no 'eye' node is present in this shape's mesh, the position of the shape's centroid is returned instead.

See Also
getEyeTransform, getEyeVector

getEyeTransform()

Purpose
Use the ***getEyeTransform*** method to get the transform of this shape's 'eye' node.

Returns
Returns a transform, "posX posY posZ rotX rotY rotZ rotTheta", representative of a shape's eye postion, orientation vector, and the rotation about that vector in radians. If not 'eye' node is present in this shape's mesh, the transform of the shape itself is returned.

See Also
getEyePoint, getEyeVector

getEyeVector()

Purpose
Use the ***getEyeVector*** method to retrieve a vector representing the direction the shape's 'eye' node is facing.

Returns
Returns a vector representing the direction a shape is looking. If this shape's mesh contains no 'eye' node, the shape's forward vector will be returned instead.

Notes
Unless this shape is being observed through (i.e. the camera is hooked up to this shape and using its transforms for viewing), this will default to getForwardVector().

See Also
getEyePoint, getEyeTransform

Image/Weapons Specifics

getImageAmmo	getImageLoaded	getImageSkinTag	getImageState
getImageTrigger	getMuzzlePoint	getMuzzleVector	getPendingImage
isImageFiring	setImageAmmo	getImageLoaded	setImageTrigger

getImageAmmo(*slot*)

Purpose
Use the ***getImageAmmo*** method to determine if the image mounted at ***slot*** has ammo.

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.

Returns
Returns true if there is an image mounted at slot and it has ammo, false otherwise.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
setImageAmmo

getImageLoaded(*slot*)

Purpose
Use the ***getImageLoaded*** method to determine if the image mounted at ***slot*** is loaded.

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

Returns
Returns true if there is an image mounted at slot and it is loaded, false otherwise.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
setImageLoaded

getImageSkinTag(*slot*)

Purpose
Use the ***getImageSkinTag*** method to get the skin tag for the image mounted at slot on this shape.

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.

Returns
Returns an integer representing the tag number for the texture the image mounted at slot is using.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.
Skin tags are selected when mounting an image to a shape.

See Also
mountImage

getImageState(*slot*)

Purpose
Use the ***getImageState*** method to determine state name that an image mounted in ***slot*** is at. i.e. If an image is mounted in ***slot***, what state is its state-machine currently in?

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.

Returns
Returns a string representing the state name that the image mounted at ***slot*** is currently at. If there is not state machine defined for the image at ***slot***, or if there no image at ***slot***, or if ***slot*** is > 7, this method returns "Error".

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

getImageTrigger(*slot*)

Purpose
Use the ***getImageTrigger*** method to get the trigger value for the image mounted at slot.

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.

Returns
Returns a boolean value specifying whether the trigger for the image at ***slot*** is active (true), or inactive (false). If no image is mounted at ***slot*** or if ***slot*** is > 7, this method returns 0.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
setImageTrigger

getMuzzlePoint(*slot*)

Purpose
Use the ***getMuzzlePoint*** method to get the position of the 'muzzlePoint' node for the image mounted at slot.

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

Returns
If an image is mounted at ***slot***, and if it specifies the node 'muzzlePoint', this method will return the XYZ position of the node. If there is an image mounted at ***slot***, and that image does not contain a 'muzzlePoint' node, this method returns the position of the image's centroid. In all other cases, this method returns “0 0 0”.

See Also
getMuzzleVector

getMuzzleVector(*slot*)

Purpose
Use the ***getMuzzleVector*** method to get the direction the 'muzzlePoint' node for the image mounted at slot is pointing.

Syntax
slot - An integer value between 0 and 7 representing a mount slot on this shape.

Returns
If an image is mounted at ***slot***, and if it specifies the node 'muzzlePoint', this method will return the XYZ pointing vector of the node. If there is an image mounted at ***slot***, and that image does not contain a 'muzzlePoint' node, this method returns the equivalent of a forward vector for the weapon itself. In all other cases, this method returns "0 0 0".

The returned vector can be modified by setting the *image.correctMuzzleVector* field. If this value is set to true, the vector will point from its origin to the position the player's eye is looking, otherwise, it will point in the true direction the weapon or weapon's muzzlePoint is facing, which may not be exactly where the player is looking.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
getMuzzlePoint

getPendingImage(*slot*)

Purpose
Use the ***getPendingImage*** method to determine if any images are pending for the specified slot.

Syntax
slot - An integer value between 0 and 7 representing a mount slot on this shape.

Returns
Returns 0 if no images are pending or an integer representing the image datablock ID if an image is pending.

Notes
A pending image is an image that is waiting to mount a slot but is prevented from doing so by the presence of another 'blocking' image.

Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

isImageFiring(***slot*** **)**

Purpose
Use the ***isImageFiring*** method to determine if this image's state machine is currently in a state with the stateFire field set to true.

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.

Returns
Returns true if this image's state machine is currently in a state with the stateFire field set to true, false otherwise.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
getImageState

setImageAmmo(***slot*** **,** ***hasAmmo*** **)**

Purpose
Use the ***setImageAmmo*** method to give an image in ***slot*** ammo, or to remove ammo from an image in ***slot***.

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.
hasAmmo – A boolean value. If set to true, this image is set has having ammo, otherwise it is set as not having ammo.

Returns
Returns true on success, and false on failure.

Notes
Weapons support both 'ammo' and 'loaded' as distinct concepts.

Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
setImageLoaded

setImageLoaded(***slot*** **,** ***loaded*** **)**

Purpose
Use the ***setImageLoaded*** method to set an image in ***slot*** as loaded or unloaded.

Syntax
   slot – An integer value between 0 and 7 representing a mount slot on this shape.
hasAmmo – A boolean value. If set to true, this image is set has being loaded, otherwise it is set as not being loaded.

Returns
Returns true on success, and false on failure.

Notes
Weapons support both 'ammo' and 'loaded' as distinct concepts.

Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
setImageAmmo

setImageTrigger(***slot*** **,** ***triggered*** **)**

Purpose
Use the ***setImageTrigger*** method to set the trigger state for an image in slot.

Syntax
     slot – An integer value between 0 and 7 representing a mount slot on this shape.
triggered – A boolean value. If set to true, this image is triggered (firing), otherwise it is un-triggered (not firing).

Returns
Returns true on success, and false on failure.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

Velocity

getVelocity()	setVelocity()

getVelocity()

Purpose
Use the ***getVelocity*** method to get the current velocity of this shape.

Returns
Returns an XYZ vector equivalent to the magnitude and direction of this shape's movement.

See Also
applyImpulse, setVelocity

setVelocity(*velocity*)

Purpose
Use the ***setVelocity*** method to set this shape's current velocity.

Syntax
veclocity – An XYZ vector equivalent to the new magnitude and direction of this shape's movement.

Returns
Returns true on success, false on failure.

Notes
It is sometimes nicer to use this method than *applyImpluse* because with this we can ignore a shape's mass while with *applyImpluse* mass directly affects the end velocity of the shape.

See Also
appyImpulse, getVelocity

Impulses

applyImpulse

applyImpulse(*position* , *impulseVector*)

Purpose
Use the ***applyImpulse*** method to apply an instantaneous acceleration to a shape at world ***position***.

Syntax
position – An XYZ vector specifying the position at which to apply the impulse.
impulseVector – An XYZ vector encoding the magnitude and direction of the impulse.

Returns
No return value.

Notes
To apply an impulse, the shape must have a non-zero positive mass, or applying an impulse will crash the engine. Also, applying an impulse whose magnitude is >= about 40 times the mass of a shape may cause the engine to lock up temporarily.

See Also
setVelocity

Camera Settings

getCameraFOV **setCameraFOV**

getCamerFOV( )

Purpose
Use the ***getCamerFOV*** method to get the current field-of-view FOV.

Returns
Returns the current FOV setting, a value between 0.0 and 180.0

See Also
setCameraFOV

setCamerFOV(*FOV*)

Purpose
Use the ***setCamerFOV*** method to set the current field-of-view (FOV) if this is a camera shape, or if a camera shape is using this shape to get it's FOV setting.

Syntax
FOV – Field-of-view. A floating-point value between 0.0 and 180.0.

Returns
No return value.

Notes
Changing the FOV will make the view zoom in and out, depending on the setting. i.e. If we start at an default FOV of 90.0 and increase it to 270.0, the scene will zoom out. If we change it to 35.0 it will zoom in.

See Also
getCameraFOV

Animations

pauseThread	playThread	setThreadDir	stopThread

pauseThread(*thread*)

Purpose
Use the ***pauseThread*** method to pause a currently playing animation.

Syntax
thread – An integer value between 0 and 3 specifying a previously started animation sequence.

Returns
Returns true if there was a previous thread playing, false otherwise.

See Also
playThread, stopThread

playThread(*thread* , *sequenceName*)

Purpose
Use the ***playThread*** method to play a new sequence specified by ***sequenceName*** in the animation ***thread***.

Syntax
thread – An integer value between 0 and 3 specifying a thread/slot to start this animation in.
sequenceName – A valid animation name for the mesh this shape is using.

Returns
Returns true if the thread was successfully started, otherwise returns false.

Notes
Playing an new animation in a ***thread*** that already has an active animation will stop the active animation and reset affected nodes to their pre-animation positions.

See Also
pauseThread, setThreadDir, stopThread

setThreadDir(*thread* , *forward*)

Purpose
Use the ***setThreadDir*** method to set the direction a specific ***thread*** should be played in.

Syntax
thread – An integer value between 0 and 3 specifying a previously started animation sequence.
forward – A boolean value. If set to true, the animation will play forward, otherwise it will play in reverse.

Returns
Returns true if the direction could be set for the specified ***thread***, otherwise returns false.

Notes
A thread needs to have already been started for this to work, but after that this will work on any sequence whether it be already completed, or a cyclic animation that never ends.

See Also
pauseThread, playThread, stopThread

stopThread(*thread*)

Purpose
Use the ***stopThread*** method to stop a previously started sequence from playing.

Syntax
thread – An integer value between 0 and 3 specifying a previously started animation sequence.

Returns
Returns true if the thread was successfully stopped, otherwise returns false.

See Also
pauseThread, playThread, setThreadDir

Sound

playAudio	**stopAudio**

playAudio(*thread* , *audioProfile*)

Purpose
Use the ***playAudio*** method to play a sound specified by ***audioProfile*** in the slot specified by ***thread***.

Syntax
thread – An integer value between 0 and 3 specifying a slot to play the sound in.
audioProfile – An audioprofile datablock previously created using the datablocks keyword, not the new keyword.

Returns
Returns true if the sound was successfully started, otherwise returns false.

Notes
Be sure to only use audioProfiles made using the datablock keyword or the sounds will not play on remote clients.

See Also
stopAudio

stopAudio(*thread*)

Purpose
Use the ***stopAudio*** method to stop a previously started audio ***thread***.

Syntax
thread – An integer value between 0 and 3 specifying a slot in which a sound was previously started.

Returns
Returns true if the sound was successfully stopped, otherwise returns false.

See Also
playAudio

Mounting

getMountedImage	getMountedObjectCount	getMountedObjectNode
getMountNodeObject	getMountSlot	getMountedObject
getObjectMount	getPendingImage	getSlotTransform
isImageMounted	isMounted	mountImage
mountObject	unmount	unmountImage
unmountObject		

getMountedImage(*slot*)

Purpose
Use the ***getMountedImage*** method to get the datablock ID of the image mounted in ***slot***.

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.

Returns
Returns the datablock ID of the image mounted in ***slot***, or zero if no image is mounted.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
isImageMounted, mountImage, unmountImage

getMountedObjectCount()

Purpose
Use the **getMountedObjectCount** method to determine how many objects are currently mounted to this shape.

Returns
Returns an integer between 0 and 8 representing the number of objects that are mounted to this shape.

Notes
A shape may have at most eight objects mounted to it at any one time.

This method does not count images that are mounted to the shape.

See Also
mountObject, unmountObject

getMountedObjectNode(*slot*)

Purpose
Use the ***getMountedObjectNode*** method to determine the mount node index of an object mounted in **slot.**

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.

Returns
Returns a numeric value between 0 and 31 if an object is mounted is slot, otherwise returns -1.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
getMountNodeObject

Return the mount node used by the object/image mounted at ***slot***.

getMountNodeObject(*node*)

Purpose
Use the ***getMountNodeObject*** method to get the ID of the object mounted at mount ***node***.

Syntax
node – A node value between 0 and 31 representing a mount node in the mesh used
       by this shape.

Returns
Returns an integer value equal to the ID of the object mounted on ***node***. If not object is mounted at node, this method returns -1.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
getMountedObjectNode

getMountSlot(*imageHandle*)

Purpose
Use the ***getMountSlot*** method to determine what slot the image datablock specified by ***imageHandle*** is mounted in.

Syntax
imageHandle – The name of ID of a ShapeBaseImageData datablock.

Returns
Returns a value between 0 and 7 if ***imageHandle*** is mounted to this shape, or -1 if it is not.

getMountedObject(*slot*)

Purpose
Use the ***getMountedObject*** method to get the ID of the object mounted at mount ***slot***.

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.

Returns
Returns an integer value equal to the ID of object mounted is ***slot***. If not object is mounted in this slot, the method returns -1.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
getMountedObjectNode

getObjectMount()

Purpose
Use the ***getObjectMount*** method to get the ID of the object this object is mounted to.

Returns
Returns an integer value representing the ID of an object that this shape is mounted to, or 0 if this object is not mounted to another object.

getPendingImage(*slot*)

Purpose
Use the ***getPendingImage*** method to get the ID of an image that is pending for ***slot***.

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.

Returns
Returns an integer value representing the ID of a shapeBaseImageData object that is pending for ***slot***. If no image is pending, returns 0.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

getSlotTransform(*slot*)

Purpose
Use the ***getSlotTransform*** method to get the transform the specified ***slot***.

Syntax
slot – An integer value between 0 and 7 representing a mount slot on this shape.

Returns
Returns a seven-element floating-point vector representing the transform of the node that an object tracked in ***slot*** is mounted to. i.e. Once we have a shape mounted to another shape, we can determine what slot the shape is tracked in and then get the transform for the node that is being used for the mount.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

isImageMounted(*imageHandle*)

Purpose
Use the ***isImageMounted*** method to determine if the shapeBaseImageData specified by ***imageHandle*** is mounted on this shape.

Syntax
imageHandle – The datablock ID of a shapeBaseImageData object to check for.

Returns
Returns true if the specified image is mounted to this shape, otherwise returns false.

isMounted()

Purpose
Use the ***isMounted*** method to see if this object is mounted to another shape.

Returns
Returns true if this shape is mounted to another shape, otherwise returns false.

mountImage(*imageHandle, slot* [, *loaded* [, *skinName*]])

Purpose
Use the ***mountImage*** method to mount the image specified by ***imageHandle*** to this image, using ***slot***. Optionally, this image can be set to the '***loaded***' state and given an alternate ***skinName***.

Syntax

```
imageHandle – The ID or name of a valid shapeBaseImageData datablock.
       slot – An integer value between 0 and 7 representing a mount slot on
              this shape.
     loaded – A boolean value.  If set to true, the image the image's loaded
              state will be true.
   skinName – A optional skin tag used to specify a specialized 'team' skin for
              the image.
```

Returns
Returns true on success, false otherwise.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

Don't forget. Images specify the node that the mount to in their 'mountPoint' field.

See Also
mountObject

mountObject(*objectHandle* , *node*)

Purpose
Use the ***mountObject*** method to mount the shape specified by ***objectHandle*** to this shape at mount ***node***.

Syntax

```
objectHandle – The ID or name of the shape to mount to this shape.
        node – The mount node on this shape's mesh upon which to mount the shape.
```

Returns
Returns true on a successful mount, otherwise returns false.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
mountImage

unmount()

Purpose
Use the ***unmount*** method to force this object to be unmounted from a shape it is mounted to.

Returns
No return value.

See Also
unmountImage, unMountObject

unmountImage(***slot*** **)**

Purpose
Use the ***unmountImage*** method to unmount an image mounted to this shape in ***slot***.

Syntax
slot - An integer value between 0 and 7 representing a mount slot on this shape.

Returns
Returns true if the image was unmounted successfully, otherwise returns false if no image was present in that ***slot***. Images cannot 'resist' unmounting.

Notes
Do not confuse slots with mount nodes. A shape can have up to 32 mount nodes, but only has eight mount slots. Mount nodes are points on the mesh used by the shape while slots are used to track how many objects and images are mounted to a shape.

See Also
unmountObject

unmountObject(***objectHandle*** **)**

Purpose
Use the ***unmountObject*** method to cause the object specified by objectHandle to unmount from this shape.

Syntax
objectHandle - The ID or name of an object mounted to this shape.

Returns
Returns true if ***objectHandle*** is in fact mounted to this shape and is then unmounted by this method. Returns false if ***objectHandle*** is not mounted to this shape.

See Also
unmountImage

getControllingClient()	getControllingObject()

getControllingClient()

Purpose
Use the **getControllingClient** method to get the ID of the client controlling this shape.

Returns
Returns the ID of the GameConnection (client) controlling this shape, or 0 if no client is controlling this shape.

See Also
getControllingObject

getControllingObject()

Purpose
Use the **getControllingObject** method to get the ID of the object controlling this object. Shapes may controlled by other shapes by having the controlling shape pass commands sent to it from the client to a surrogate object instead. i.e. Object A can be controlled by the client and send movement inputs to object B, in effect controlling object B.

Returns
Returns the ID of the shape controlling this object, otherwise if no object is controlling this shape returns 0.

The playe class has methods for setting up control of other shapes.

A.2.59. ShapeBaseData

Fields

Field Name	Description	Sample or Range
Rendering		
cloakTexture	Path to texture to be used when shape is cloaked.	See lesson samples.
emap	A boolean value specifying whether to render environmental map or not.	false
shapeFile	Path to DTS file containing the model for this shape.	See lesson samples.
Physics		
density	The density of this shape. Affects whether this shape will sink or float in water (relative to water density).	See lesson samples.
drag	Drag presented by this shape as it moves through the air.	0
mass	Mass for this shape.	100
Damage		
debris	DebrisData datablock to use for post-destruction debris.	See lesson samples.
debrisShapeName	Path to DTS file containing the model for this shape which is to be rendered post destruction.	See lesson samples.

Field Name	Description	Sample or Range
explosion	ExplosionData datablock to use for an explosion when this shape is destroyed	See lesson samples.
isInvincible	If this boolean value is true, this shape does not take damage.	false
maxDamage	Maximum damage this shape can accrue.	100
renderWhenDestroyed	Boolean value specifying whether to render this shape or not once it's damageState is 'Destroyed'.	false
underwaterExplosion	ExplosionData datablock to use for an explosion when this shape is destroyed and submerged in water.	See lesson samples.
Damage Reference Values	These values do not change engine functions and are meant to be used by scripts.	
destroyedLevel	Damage level at which this shape will be considered destroyed.	100
disabledlevel	Damage level at which this shape will be considered disabled.	50
repairRate	Rate (in points per tick) at which this shape is repaired when current damage > 0.	0.03125 (== 1 point per second)
Energy		
inheritEnergyFromMount	When mounted, use mount's energy.	false
maxEnergy	Maximum energy this object can have.	50
Energy Reference Values	These values do not change engine functions and are meant to be used by scripts.	
rechargeRate	Rate (in points per tick) at which this shape is recharged when current energy < maxEnergy.	0.03125 (== 1 point per second)
Camera Settings		
cameraDefaultFOV	Initial field of view for the camera.	[0.0 , 180]
cameraMaxDist	Maximum distance from camera to observing object.	[cameraMinDist , inf)
cameraMaxFOV	Maximum field of view for the camera.	[0.0 , 180]
cameraMinDist	Minimum distance from camera to observing object.	[0.0 , cameraMaxDist]
cameraMinFOV	Minimum field of view for the camera.	[0.0 , 180]
firstPersonOnly	Render shape when in first person (only applies to shapes that are observed through.	[false , true]
observeThroughObject	If true, camera will use this shape's camera parameters, when this is the controlled object	[false , true]
useEyePoint	Use the eye node specified by this shape instead.	[false , true]
Miscellaneous		
aiAvoidThis	A boolean hit to be used by AI scripts to determine if an AI agent should 'avoid' this object. Scripts are responsible for dealing with this.	[false , true]
computeCRC	Calculate a cyclic-redundancy-code for this shape.	[false , true]

Console Method Summaries

checkDeployPos	getDeployTransform

Console Methods

Deployment Helpers

getDeployTransform(*position* , *normal*)

Purpose
Use the **getDeployTransform** method to retrieve the transform that would result from deploying an unscaled object using this datablock at ***position*** with the specified normal.

Syntax
position - The postion to create the transform for.
normal - A vector representing the object's UP vector. This can be thought of as an object's tilt or leaning vector. If the object is to be placed at an angle, pass in a unit-vector that has the same orientation as a vector starting at ***position*** and aligning with the required angle.

Returns
Returns a deployment (placement) transform for an un-scaled and un-rotated datablock derived shape placed at the required location with an optional tilt.

See Also
checkDeployPos

checkDeployPos(*transform*)

Purpose
Use the **checkDeployPos** method to determine if placing an unscaled version of an object using this datablock will cause that shape to be embedded in a staticShape or an InteriorInstance.

Syntax
transforms - A normal transform, or a transform calculated using *getDeployTransform*.

Returns
Returns false if the placement will result in the new shape being embedded in a staticShape or an InteriorInstance. Otherwise, it will return true.

Notes
This check doesn't examine the area for all possible object, just staticShapes and Interiors. To include more objects you will need to edit the engine and add them to the list of 'interesting' objects.

See Also
getDeployTransform

A.2.60. ShapeBaseImageData

ShapeBaseImageData has no associated class as other xyzData classes do. This special class is used to represent geometry that can only be mounted to an existing ShapeBase object. ShapeBaseImageData objects cannot otherwise be instantiated like other Shape objects.

Fields

Field Name	Description	Sample or Range
Rendering		
emap	A boolean value specifying whether to render environmental map or not.	--
firstPerson	If true, render this image in 1st POV.	--
shapeFile	Path to DTS file containing the model for this shape.	--
Projectile		
projectile	ProjectileData datablock	--
Cloaking		
cloakable	A boolean value specifying whether this image cloaks when the shape it is mounted to is cloaked.	--
Mounting		
eyeOffset	Rendering position offset (only affects 1st POV).	--
eyeRotation	Rendering rotation offset (only affects 1st POV).	--
mountPoint	Named mount node to mount to on the receiving shape.	[0 , 31]
offset	Position offset from mount point.	--
rotation	Rotation offset from mount point.	--
Lighting		
lightColor	Three-element floating-point vector specifying RGB components of light.	"r g b a" In range [0.0 , 1.0]
lightRadius	Floating-point value specifying radius for light emission.	[0.0 , 20.0]
lightTime	Integer value specifying time (in milliseconds) for light to pulse on-off-on.	[0 , inf)
lightType	String specifying type of light does this image emits.	"NoLight" "ConstantLight" "PulsingLight"
Physics		
mass	Floating-point value specifying mass for this image.	--
Weapons		
accuFire	Has no effect.	--
casing	DebrisData datablock to use for ejected casings.	
correctMuzzleVector	Boolean value specifying that the muzzle vector should be calculated from the players eyeVector, not from the muzzleVector of the image.	[false , true]
minEnergy	Floating-point value specifying minimum energy required to 'fire' this weapon.	--
shellExitDir	Ejection vector for ejected casings.	--
shellExitVariance	Floating-point value specifying variance in ejection vector direction.	[0.0 , 180.0]
shellVelocity	Floating-point value specifying shell ejection velocity.	[0.0 , inf]
usesEnergy	Boolean value specifying that this weapon uses energy.	[false , true]

Field Name	Description	Sample or Range
State Machine		
stateAllowImageChange[]	If false, will temporarily block other images from mounting while the state machine is executing the tasks in this state.	[false , true]
stateDirection[]	Direction of animation. **true** is forward.	[false , true]
stateEjectShell[]	Eject shell in this state.	[false , true]
stateEmitter[]	Play this particle emitter (at muzzle point or specified node).	
stateEmitterNode[]	Integer value specifying node to attach emitter to. Default is to mount to named node 'muzzlepoint'	
stateEmitterTime[]	Floating-point value specifying time to run emitter.	
stateEnergyDrain[]	Floating-point value specifying	
stateFire[]	First state with this a true is the state entered by the client when it receives the 'fire' event.	
stateIgnoreLoadedForReady[]	If set to true, and both ready and loaded transitions are true, the ready transition will be taken instead of the loaded transition.	[false , true]
stateLoadedFlag[]	Set loaded state of shape to specified value.	"Ignore" "Loaded" "Empty"
stateName	Name of this state.	
stateRecoil[]	Play specified recoil animation.	"NoRecoil" "LightRecoil" "MediumRecoil" "HeavyRecoil"
stateScript[]	Method to execute on entering this state. Scoped to this image class name, then shapeBaseImageData, ...	
stateSequenceRandomFlash[]	Play random flash animation.	[false , true]
stateSequence[]	Play this animation.	
stateSound[]	Play sound specified by this audio file.	
stateSpinThread[]	Play this spin animation (blended).	"Ignore" "Stop" "SpinUp" "SpinDown" "FullSpeed"
stateTimeoutValue[]	Time in seconds for this state to time out.	[0.0 , inf)
stateTransitionOnNotLoaded[]	Transition to this named state if the state of loaded is 'Empty'.	someStateName
stateTransitionOnTriggerDown[]	Transition to this named state if the fire button is pressed.	someStateName
stateTransitionOnTriggerUp[]	Transition to this named state if the fire button is released.	someStateName
stateTransitionOnAmmo[]	Transition to this named state when the ammo the image has ammo.	someStateName
stateTransitionOnLoaded[]	Transition to this named state when the loaded state of the image is 'Loaded'.	someStateName
stateTransitionOnNoAmmo[]	Transition to this named state when the the image has no ammo.	someStateName
stateTransitionOnNoTarget[]	Transition to this named state when the image has no target.	someStateName
stateTransitionOnNotWet[]	Transition to this named state when the image is not underwater.	someStateName
stateTransitionOnWet[]	Transition to this named state when image is underwater.	someStateName
stateTransitionTarget[]	Transition to this named state when the image has a target.	someStateName
stateTransitionTimeout[]	Transition to this named state when the current state transition timeout time has elapsed.	someStateName
stateWaitForTimeout[]	If false, this state ignores timeout and transitions immediately if other tests are met.	[false , true]

Field Name	Description	Sample or Range
Miscellanous		
computeCRC	Verify that the CRC of the client's image matches the server's CRC for the image on load by client.	[false , true]

A.2.61. SimDataBlock

This class is the root class for all datablock classes. It does not define any fields or console methods of it's own.

A.2.62. SimGroup

Same fields as SimSet.

A.2.63. SimObject

Console Method Summaries

delete	**dump**	**getClassName**
getGroup	**getId**	**getName**
getType	**save**	**schedule**

Console Methods

delete()

Purpose
Use the ***delete*** method to delete this object.

Returns
No return value.

Notes
When an object is deleted, it automatically:
- Unregisters its ID and name (if it has one) with the engine.
- Removes itself from any SimGroup or SimSet it may be a member of.
- (eventually) returns the memory associated with itself and its non-dynamic members.
- Cancels all pending %obj.schedule() events.

For objects in the GameBase, ScriptObject, or GUIControl hierarchies, an object will first:
- Call the onRemove() method for the object's namespace.

dump()

Purpose
Use the ***dump*** method to display the following information about this object:

- All engine registered console methods (including parent methods) for this object.
- All script registered console methods (including parent methods) for this object.
- All Non-Dynamic Fields
- All Dynamic Fields

Returns
No return value.

getClassName()

Purpose
Use the ***getClassName*** method to get the ConObject class name of this object.

Returns
Returns a string containing the ConObject registered name of this object's class.

Some Possible Returns:
- SimObject
- WheeledVehicle
- Player
- ...

See Also
getType

getGroup()

Purpose
Use the ***getGroup*** method to determine if this object is contained in a SimGroup and if so, which one.

Returns
Returns the ID of the SimGroup this shape is in or zero if the shape is not contained in a SimGroup.

getId()

Purpose
Use the ***getId*** method to get the numeric ID of this shape.

Returns
Returns the unique numeric ID of this shape.

See Also
getName, setName

getName()

Purpose
Use the ***getName*** method to get the name (if any) for this shape.

Returns
Returns a string containing the name of this shape, or the NULL string if this shape was never given a name (either when created or subsequently).

See Also
getID, setName

getType()

Purpose
Use the ***getType*** method to get the type for this object. This type is an integer value composed of bitmasks. For simplicity, these bitmasks are defined in the engine and exposed for our use as global variables.

Returns
Returns a bit mask containing one or more set bits.

Notes
To simplify the writing of scripts, a set of globals has been provided containing the bit setting for each class corresponding to a particular type. For a complete list of the bit masks, see the 'Shape Type Bitmasks' table.

```
-$TypeMasks::GameBaseObjectType
-- $TypeMasks::EnvironmentObjectType
-- $TypeMasks::ExplosionObjectType
-- $TypeMasks::ProjectileObjectType
-- $TypeMasks::ShapeBaseObjectType
--- $TypeMasks::CameraObjectType
--- $TypeMasks::ItemObjectType
--- $TypeMasks::MarkerObjectType
--- $TypeMasks::PlayerObjectType
--- $TypeMasks::StaticShapeObjectType
--- $TypeMasks::VehicleObjectType
-- $TypeMasks::TriggerObjectType
- $TypeMasks::InteriorObjectType
- $TypeMasks::StaticObjectType
- $TypeMasks::TerrainObjectType
- $TypeMasks::VehicleBlockerObjectType
- $TypeMasks::WaterObjectType
```

Two interesting general masks are:
- $TypeMasks::EnvironmentObjectType – Matches sky, sun, lightning, particle emitter nodes.
- $TypeMasks::StaticObjectType – Matches – fxFoliageReplicator, fxLight, fxShapeReplicator, fxSunlight, interiorInstance, lightning, mirrorSubObject, missionMarker, staticShape, terrain, tsStatic

See Also
getClassName

$TypeMasks::StaticObjectType	**$TypeMasks::EnvironmentObjectType**
$TypeMasks::TerrainObjectType	**$TypeMasks::InteriorObjectType**
$TypeMasks::WaterObjectType	**$TypeMasks::TriggerObjectType**
$TypeMasks::MarkerObjectType	**$TypeMasks::GameBaseObjectType**
$TypeMasks::ShapeBaseObjectType	**$TypeMasks::CameraObjectType**
$TypeMasks::StaticShapeObjectType	**$TypeMasks::PlayerObjectType**
$TypeMasks::ItemObjectType	**$TypeMasks::VehicleObjectType**
$TypeMasks::VehicleBlockerObjectType	**$TypeMasks::ProjectileObjectType**
$TypeMasks::ExplosionObjectType	

Shape Type Bitmasks

schedule(*time* , *command* , *<arg1 ... argN>*)

Purpose
Use the ***schedule*** method to schedule an action to be executed upon this object ***time*** milliseconds in the future.

Syntax

time –	Time in milliseconds till action is scheduled to occur.
command –	Name of the command to execute. This command must be scoped to this object (i.e. It must exist in the namespace of the object), otherwise the schedule call will fail.
arg1...argN –	These are optional arguments which will be passed to ***command***. This version of schedule automatically passes the ID of %obj as arg0 to ***command***.

Returns
Returns an integer schedule ID.

Notes
The major difference between this and the schedule console function is that if this object is deleted prior to the scheduled event, the event is automatically canceled.

times should not be treated as exact since some 'simulation delay' is to be expected. The minimum resolution for a scheduled event is ~32 ms, or one tick.

schedule does not validate the existence of ***command***. i.e. If you pass an invalid console method name, the schedule() method will still return a schedule ID, but the subsequent event will fail silently.

See Also
See the schedule console function and its corresponding helper functions.

setName(*name*)

Purpose
Use the ***setName*** method to give this shape a new name.

Syntax
name - A string containing the new name for this shape, or a NULL string to un-name this shape.

Returns
No return value.

Notes
Names can be compose of non-alphanumeric symbols, but be careful when doing this.

See Also
getID, getName

A.2.64. SimSet

This is a generic container class for SimObjects. It provides a basic set of console methods for storing and accessing SimObjects contained in the set. Any SimObject may be contained in multiple SimSets

Console Methods

add	bringToFront	clear	getCount
getObject	isMember	listObjects	pushToBack
remove			

add(*obj1* , *...*)

Purpose
Use the ***add*** method to add one or more objects to this SimSet.

Syntax
```
obj1 - The ID or name of an object to add to the SimSet.
 ... - A comma separated list of as many names and object IDs as you wish,
       all of which will be added to this SimSet.
```

Returns
No return value.

See Also
clear, bringToFront, getCount, getObject, isMember, listObjects, pushToBack.

Add new object to SimSet. Objects are validated on addition. i.e. If you pass an invalid ID an error will be printed and the add will fail.

obj1 - A object ID to add to the SimSet.
... - A list of comma separated IDs may be passed.

bringToFront(object)

Purpose
Use the ***bringToFront*** method to move ***object*** to front of SimSet (queue). This is useful for sorting and later when this class is used as the base to the GUIControl class.

Syntax
object – The ID or name of an object already in the set.

Returns
No return value.

Notes
Do not attempt to call this for objects that are not in the SimSet as you can crash the engine.

See Also
pushToBack

clear()

Purpose
Use the ***clear*** method to remove all entries from SimSet.

Notes
This does not cause the entries to be deleted.

See Also
add, remove

getCount()

Purpose
Use the ***getCount*** method to determine how many objects are currently being tracked by this SimSet.

Returns
Returns an integer value between 0 and inf equal to the number of objects currently being tracked by this SimSet.

See Also
add, clear, remove

getObject(index)

Purpose
Use the ***getObject*** method to get the ID of an object stored in this SimSet at position ***index***.

Syntax
index – The queue position of the object to look for/get.

Returns
Returns ID of object at ***index*** in SimSet. If no object is found at that index, 0 is returned.

isMember(object)

Purpose
Use the ***isMember*** method to determine if ***object*** is being tracked by this SimSet.

Syntax
object – An object name or ID to check for in this set.

Returns
Returns true if ***object*** is in this SimSet, otherwise returns false.

Notes
Unless you are absolutely sure about membership be sure to use this method prior to using *bringToFront* or *pushToBack*.

See Also
bringToFront, pushToBack

listObjects()

Purpose
Use the ***listObjects*** method to dump a list of all objects in this SimSet to the console.

Returns
No return value.

Notes
This is a helpful debug tool.

pushToBack(object)

Purpose
Use the ***pushToBack*** method to push an ***object*** (already in the SimSet) to then back of the queue.

Syntax
object – The ID or name of an object already in the set.

Returns
No return value.

Notes
Do not attempt to call this for objects that are not in the SimSet as you can crash the engine.

See Also
bringToFront

remove(*obj1 , ...*)

Purpose
Use the ***remove*** method to remove one or more objects from this SimSet.

Syntax
obj1 – The ID or name of an object already in the SimSet.
... – A comma separated list of objects already in this SimSet.

Returns
No return value.

Notes
Attempting to remove an object from this SimSet which is not in the SimSet will have no effect.

See Also
add, clear

A.2.65. Sky

Fields

Field Name	Description	Sample or Range
cloudHeightPer0	Relative height of layer 0 clouds.	[0.0 , cloudHeightPer1]
cloudHeightPer1	Relative height of layer 1 clouds.	[cloudHeightPer0 , cloudHeightPer2]
cloudHeightPer2	Relative height of layer 2 clouds.	[cloudHeightPer1 , 1.0]
cloudSpeed1	Cloud scrolling vs. windVelocity multiplier for layer 0 clouds.	[0.0 , inf.0)
cloudSpeed2	Cloud scrolling vs. windVelocity multiplier for layer 1 clouds.	[0.0 , inf.0)
cloudSpeed3	Cloud scrolling vs. windVelocity multiplier for layer 2 clouds.	[0.0 , inf.0)
cloudText[0,2]	--	Not used
fogColor	Color of general fog.	"1.0 0.5 0.5 1.0"
fogDistance	Affects fog density as a ratio of visibleDistance.	[0.0 , inf.0)
fogStorm1	Enables fog storm fading of layer 0.	[false , true]
fogStorm2	Enables fog storm fading of layer 1.	[false , true]
fogStorm3	Enables fog storm fading of layer 2.	[false , true]
fogVolume1	Layer 0 fog definition.	"dist low-elev high-elev"
fogVolume2	Layer 1 fog definition.	"dist low-elev high-elev"
fogVolume3	Layer 2 fog definition.	"dist low-elev high-elev"
fogVolumeColor[1:3]	--	Not used
materialList	Path to sky DML file.	dml file path
noRenderBans	If set to true, the sky box is not allowed to be fogged out.	[false , true]
renderBottomTexture	If set to true, the bottom texture in the DML file will be rendered.	[false , true]
SkySolidColor	The color for the untextured sky box.	"r g b a" (float)
useSkyTextures	Enables sky box texture rendering.	[false , true]
visibleDistance	Maximum render distance.	[0.0 , inf)
windEffectPrecipitation	If set to true, wind will blow precipitation.	[false , true]
windVelocity	Magnitude and direction of wind.	“1.5 2.0 0.0”

Globals

Variable Name	Description	Sample or Range
pref::CloudOutline	Enable or disable cloud outlining.	[false , true]
pref::CloudsOn	Enables cloud rendering.	[false , true]
pref::NumCloudLayers	Limits number of cloud layers that are rendered.	[0 , 3]
pref::SkyOn	Enables the sky.	[false , true]

Console Methods

getWindVelocity()	**realFog()**	**setWindVelocity()**	**stormClouds()**
stormCloudsShow()	**stormFog()**	**stormFogShow()**	

getWindVelocity()

Purpose
Use the ***getWindVelocity*** method to get the current wind velocity vector.

Returns
Returns a three-element floating point vector representing the direction and velicity of the wind.

setWindVelocity(***x*** **,** ***y*** **,** ***z*** **)**

Purpose
Use the ***setWindVelocity*** method to modify the current wind velocity.

Syntax
x – A floating-point value representing the X component of the wind velocity.
y – A floating-point value representing the Y component of the wind velocity.
z – A floating-point value representing the Z component of the wind velocity.

Returns
No return value.

stormClouds(***show*** **,** ***duration*** **)**

Purpose
Use the ***stormClouds*** method to ***show*** or hide all defined cloud layers over ***duration*** seconds.

Syntax
show – Boolean value specifying whether to show (true), or hide (false) clouds.
duration – Time in seconds within which to achieve results.

Returns
No return value.

Notes
- ***duration*** cannot be zero
- Layers fade in top-to-bottom and fade out bottom-to-top

See Also
stormCloudsShow, stormFog, stormFogShow

stormCloudsShow(show)

Purpose
Use the ***stormCloudsShow*** method to show or hide all defined cloud layers instantly.

Syntax
show – Boolean value specifying whether to show (true), or hide (false) clouds.

Returns
No return value.

See Also
stormClouds, stormFog, stormFogShow

stormFog(*percent* , *duration*)

Purpose
Use the ***stormFog*** method to show or hide all stormFog enabled fog layers over ***duration*** seconds.

Syntax
percent – Floating-point value specifying ending fade level for fog layers.
duration – Time in seconds within which to achieve results.

Returns
No return value.

Notes
- ***duration*** cannot be zero
- Layers fade in bottom to top and fade out top-to-bottom
- stormFog**n** – Must be checked for that layer to be affected by this method.
- ***percent*** – Must be in range [0.0 , 1.0]

See Also
stormClouds, stormCloudsShow, stormFogShow

stormFogShow(show)

Purpose
Use the ***stormFogShow*** method to show or hide all stormFog enabled fog layers instantly.

Syntax
show – Boolean value specifying whether to show (true), or hide (false) fog.

Returns
No return value.

See Also
stormClouds, stormCloudsShow, stormFog

A.2.66. SpawnSphere

Mission marker used to mark drop points. Can be used for many things. The fields in this object are only used by scripts, not the engine.

Fields

Field Name	Description	Sample or Range
indoorWeight	Only used by scripts.	any float
outdoorWeight	Only used by scripts.	any float
radius	Only used by scripts.	any float
sphereWeight	Only used by scripts.	any float

A.2.67. Splash

No fields or methods to discus.

A.2.68. SplashData

Fields

Field Name	Description	Sample or Range
acceleration	The acceleration of the splash, as used in the physical simulation. Affects the splash's velocity over time, along with the gravitational force acting in the system.	(-inf.0 , inf.0)
colors[0] colors[1] colors[2] colors[3]	Array of colors. Specifies what colors are to be used for splash rings at each time value specified in the *times* array field. First entry in the colors array corresponds to the initial ring color, and is interpolated with colors[1] until times[0] time has elapsed, at which time colors[1] is the draw color and it begins being interpolated with colors[2]. See the *times* and ringLifetime field documentation for more information.	"1.0 0.5 0.5"
delayMS	*Note: this field currently has no tangible effect in the engine's simulation.*	--
delayVariance	*Note: this field currently has no tangible effect in the engine's simulation.*	--
ejectionAngle	The angle at which new splash rings should be ejected, specified in degrees.	(-inf.0 , inf.0)
ejectionFreq	The frequency with which new splash rings should be created.	(-inf.0 , inf.0)
emitter[0] emitter[1] emitter[2]	Array of pointers to ParticleEmitterData datablocks which specify the particle emissions to be used for the Splash object.	see type
explosion	ExplosionData datablock, which will be used to spawn an explosion for the splash.	see type
height	*This field currently has no tangible effect in the engine's simulation.*	--
lifetimeMS	Used along with lifetimeVariance to determine the maximum time the Splash object persists in the world. Measured in whole milliseconds.	(-inf , inf)
lifetimeVariance	Used along with lifetime to determine the maximum time the Splash object persists in the world. The Splash object's actual life-time is calculated by adding the *lifetime* field with a random integer from -1*lifetimeVariance to lifetimeVariance. A Splash object will be marked as dead once it's life-time expires, but a Splash object will not delete itself until its ring life-time has expired as well; see the documentation for the	(-inf , inf)

Field Name	Description	Sample or Range
	ringLifetime field for more information. Using lifetimeVariance, separate Splash objects sharing the same SplashData datablock can be given varied behavior.	
numSegments	The number of segments used to generate each ring of the splash. Each segment has an associated splash ring texture u-coordinate, which is calculated by dividing the segment's position in the splash ring's segment list by the total number of the ring's segments, and then multiplying this result by the scalar value in the texWrap field.	(-inf , inf)
ringLifetime	The life-time, in seconds, of splash rings generated by the Splash object. Also used during the rendering of splash rings to determine the opacity of the ring; the ring starts out fully transparent, and follows a linear progression to become fully opaque at the mid-point of its life-time, after which time a linear fall-off in opacity occurs until the end of the ring's life-time, at which point it is once again fully transparent. a Splash object will not delete itself until all ring's it has generated have expired.	(-inf.0 , inf.0)
scale	*Note: this field currently has no tangible effect in the engine's simulation.*	--
soundProfile	*Note: this field currently has no tangible effect in the engine's simulation.*	--
startRadius	The inital radius with which to eject splash rings, affected by velocity over time.	(-inf.0 , inf.0)
texFactor	Scalar value used to translate the v-coordinate value of texture reads from the splash rings *texture* during their rendering. texFactor is not applied directly to scale v-coordinate reads, as the texWrap field is. Rather, texFactor is multiplied with the non-integer portion of the elapsed time (elapsedTime - int(elapsedTime)) to determine the v-coordinate texture position to be read for the splash ring.escription	(-inf.0 , inf.0)
texture	The texture file to be used for the splash rings.	~/path/filename.png
texWrap	Scalar value used along with numSegments to determine the u-coordinate value of texture reads from the splash rings *texture* during their rendering. See the numSegments field documentation for more information.	(-inf.0 , inf.0)
times[0] times[1] times[2] times[3]	Array of time values. Each entry is used to help determine the color used in rendering splash rings, as described in the *colors* field documentation.	(-inf.0 , inf.0)
velocity	The velocity of the splash, as used in physical simulation. Affects the radius of the splash over time, and the velocity of splash rings. Velocity changes over time in accordance with the value specified with the acceleration field, and the gravity affecting the physical simulation.	(-inf.0 , inf.0)
width	*Note: this field currently has no tangible effect in the engine's simulation.*	--

A.2.69. StaticShape

A concrete class derived from ShapeBase used to reprsent world objects.

Console Methods

getPoweredState()	setPoweredState()

getPoweredState()

Purpose
Use the ***getPoweredState*** method to get the shape's current 'powered' state.

Syntax
state - Is the shape powered? [false , true]

Returns
No return value.

Notes
This feature does not modify any engine behaviors and is purely for scripted use.

See Also
setPoweredState

setPoweredState(*isPowered*)

Purpose
Use the ***setPoweredState*** method to set shape's current 'powered' state.

Syntax
state - Is the shape powered? [false , true]

Returns
No return value.

Notes
This feature does not modify any engine behaviors and is purely for scripted use.

See Also
getPoweredState

A.2.70. StaticShapeData

Datablock associated with StaticShape object class.

Fields

Field Name	Description	Sample or Range
dynamicType	An integer value which, if specified, is added to the value returned by getType().	See dynamicType below
noIndividualDamage	Boolean value used as a hint to scripts, to NOT apply damage to this shape.	--

A.2.71. Sun

Mission object representing mission lighting parameters.

Fields

Field Name	Description	Sample or Range
ambient	Color / Intensity of ambient light.	"r g b i" (float)
azimuth	Azimuth of sun.	[0, 360]
color	Color / Intensity of direct light.	"r g b i" (float)
elevation	Inclination (elevation) of sun.	[0, 360]

A.2.72. TCPObject

Console Method Summaries

connect	disconnect	listen

Console Methods

connect(addr)

Purpose
Use the ***connect*** method to request a connection to a remote agent at the address ***addr***.

Syntax
addr – A string containing an address of the form: "A.B.C.D:Port", where A .. B are standard IP numbers between 0 and 255 and Port can be between 1000 and 65536.

Returns
No return value.

See Also
disconnect

disconnect()

Purpose
Use the ***disconnect*** method to close a previously opened connection without destroying the requesting TCPOpbject.

Returns
No return value.

Notes
This will close any open connection, but not destroy this object. Thus, the object can be used to open a new connection.

See Also
connect

listen(port)

Purpose
Use the ***listen*** method to allow this TCPObject to accept connections on the specified ***port***.

Syntax
port - A value between 1000 and 65536.

Returns
No return value.

send(...)

Purpose
Use the ***send*** method to send any number of parameters, as strings, one at a time to the agent at the other end of the connection.

Syntax
... - Any number of arguments, as strings. Each string is sent separately. i.e. The arguments are not concatenated.

Returns
No return value.

A.2.73. TerrainBlock

Mission object representing terrain. Default terrain in Torque measures 2 x 2 km and repeats forever. See EGTGE Volume I – Mission Objects chapter for usage.

Fields

Field Name	Description	Sample or Range
bumpOffset	Offset between bumpmap textures.	[0.0 , inf)
bumpScale	Scale of bumpmap texture.	[1 , inf)
bumpTexture	Path to texture to use for bump map textures.	--
detailTexture	Path to texture to use for terrain detailing.	--
emptySquares	List of empty squares in terrain.	--
squareSize	Vertical and horizontal distance between terrain verticies in meters. Default is 8 meters. Affects water rendering.	1 , 2 , 4 , 8 (default) , 16 , 32 , 64 , 128
terrainFile	Path to file used for terrain.	--
zeroBumpScale	Value controlling bumpmap rendering distance.	[1 , inf)
tile	Enables and disables tiling of terrain.	[false , true]

Globals

Variable Name	Description	Sample or Range
$farDistance	Distance to far render plane (where rendering stops).	Modify in sky object.
$pref::Terrain::dynamicLights		--
$pref::Terrain::enableDetails	Enable/Disable detailTexture rendering.	--
$pref::Terrain::enableEmbossBumps	Enable/Disable bump map rendering.	--
$pref::Terrain::screenError	Terrain screen error metrics. The higher this value is, the smoother terrain slopes will become (within limits). Lowering this value can increase render speed on old systems.	[0 , inf)
$pref::Terrain::texDetail	This value modifies the texturing LOD. Higher numbers equal LOWER LOD. For best results leave this at 0.	[0 , 10]
$pref::Terrain::textureCacheSize		--
$screenSize		--
$T2::dynamicTextureCount	Grand total newly rendered textures count for the terrain.	--
$T2::staticTextureCount	Grand total rendered textures count for the terrain.	--

Console Methods

getHeightfieldScript()	**getTextureScript()**	**save()**	**setHeightfieldScript()**
setTextureScript()			

These methods are used by the editors and should not be used for general scripting purposes. Please see EditorGui.cs if you want to see these method in use.

A.2.74. Trigger

Mission object representing a re-active zone in the game. This object senses the entry, exit, and presence of objects within it's bounds.

Fields

Field Name	Description	Sample or Range
polyhedron	List of (relative) coordinates representing bounds of area.	--

Console Methods

getNumObjects()	getObject()

getNumObjects()

Purpose
Use the ***getNumObjects*** method to determine how many GameBase objects are within the bounds of this trigger.

Returns
Returns an integer value specifying the number of objects that are currently within the boundaries of this trigger.

See Also
getObject

getObject(index)

Purpose
Use the ***getObject*** method to retrieve the ID of an object, at ***index***, within this trigger's object list.

Syntax
index - An integer value between 0 and numObjects, where numObjects can be retrieved with *getNumObjects*.

Returns
Returns an object ID, or 0 if no object is found at the specified index.

See Also
getNumObjects

A.2.75. TriggerData

Datablock associated with the trigger mission object.

Fields

Field Name	Description	Sample or Range
tickPeriodMS	Period in milliseconds describing time to next trigger tick. Trigger checks for current contents on each tick. Does not affect sensing of onEnter/onLeave.	[0 , inf)

A.2.76. TSShapeConstructor

Fields

Field Name	Description	Sample or Range
baseShape	The DTS file used by this shape constructor.	~/path/filename.dts
sequence[0:126]	Up to 127 sequences that can be associated with this shape constructor.	~/path/filename.dsq

A.2.77. TSStatic

Fields

Field Name	Description	Sample or Range
position	The object's placement position, not necessarily the current position.	"1.0 2.0 3.0"
rotation	The object's placement rotation, not necessarily the current rotation.	"1.0 2.0 3.0"
scale	The object's placement scale, not necessarily the current scale.	"1.0 2.0 3.0"
shapeName	DTS file for this shape	~/path/filename.dts

A.2.78. Vehicle

Fields

Field Name	Description	Sample or Range
disableMove	Used to disable a vehicle's ability to move	[false , true]

A.2.79. VehicleData

Fields

Field Name	Description	Sample or Range
bodyFriction	The vehicle's rigid body friction coefficient. Used in the physical simulation during contacts and collisions. Higher friction values dampen contact and collision forces.	(-inf.0 , inf.0)
bodyRestitution	The vehicle's rigid body restitution. Used in the physical simulation during collisions. The scalar restitution value affects the strength of the body's rebound resulting from collisions with objects. Higher restitution values yield more powerful rebound reactions.	(-inf.0 , inf.0)
cameraDecay	Scalar rate at which the third person camera offset decays, per tick.	(-inf.0 , inf.0)
cameraLag	Scalar amount by which the third person camera lags the vehicle, relative to the vehicle's linear velocity.	(-inf.0 , inf.0)
cameraOffset	The vertical offset of the vehicle's camera.	(-inf.0 , inf.0)
cameraRoll	Specifies whether the camera's rotation matrix, and the render eye transform are multiplied during camera updates.	[false , true]
collDamageMultiplier	*Note: this field currently has no tangible effect in the engine's simulation..*	(-inf.0 , inf.0)
collDamageThresholdVel	*Note: this field currently has no tangible effect in the engine's simulation.*	(-inf.0 , inf.0)
collisionTol	The minimum collision velocity required to trigger a full collision. Collisions with velocities less than the collisionTol value will be treated as collision contacts or constraints.	(-inf.0 , inf.0)
contactTol	The minimum collision velocity required to trigger a collision contact. Collisions with velocities less than the contactTol value will be treated as collision constraints.	(-inf.0 , inf.0)
damageEmitter[0] damageEmitter[1] damageEmitter[2]	Array of pointers to ParticleEmitterData datablocks which will be used to emit particles for damage effects (smoke).	see type
damageEmitterOffset[0] damageEmitterOffset[1]	Offset point at which to display damage effects.	"1.0 2.0 3.0"
damageLevelTolerance[0] damageLevelTolerance[1]	Array of floats specifying damage level thresholds. Each entry is specified as a decimal percentage of maxDamage (defined in ShapeBaseData). Each damage level is used to determine what damage effect to play.	(-inf.0 , inf.0)
dustEmitter	Array of pointers to ParticleEmitterData datablocks which will be used to emit particles at vehicle/terrain contact point.	see type
dustHeight	Height of dust effects.	(-inf.0 , inf.0)
exitingWater	The AudioProfile will be used to produce sounds when emerging from water.	see type
exitSplashSoundVelocity	The minimum velocity at which the exit splash sound will be played when emerging from water.	(-inf.0 , inf.0)
hardImpactSound	The AudioProfile used to produce sounds for hard impacts.	see type
hardImpactSpeed	Minimum speed at which the vehicle must be traveling for the hard impact sound to be played.	(-inf.0 , inf.0)
hardSplashSoundVelocity	The minimum velocity at which the hard splash sound will be played when impacting water.	(-inf.0 , inf.0)
impactWaterEasy	The AudioProfile will be used to produce sounds when a soft impact with water occurs.	see type

Field Name	Description	Sample or Range
impactWaterHard	The AudioProfile will be used to produce sounds when a hard impact with water occurs.	see type
impactWaterMedium	The AudioProfile will be used to produce sounds when a medium impact with water occurs.	see type
integration	The number of discrete steps with which to process physics data per tick.	(-inf , inf)
jetEnergyDrain	Energy drained per tick by use of the vehicle's jet, if it has one.	(-inf.0 , inf.0)
jetForce	Force generated by the vehicle's jet, if it has one. This field is only used by derived classes.	(-inf.0 , inf.0)
massCenter	The vehicle's rigid body center of mass.	"1.0 2.0 3.0"
maxDrag	Intended to clamp the maximum drag available. *Note: this field currently has no tangible effect in the engine's simulation.*	--
maxSteeringAngle	Maximum attainable steering angle, measured in radians. Steering angles are clamped to this maximum value.	(-inf.0 , inf.0)
mediumSplashSoundVelocity	The minimum velocity at which the medium splash sound will be played when impacting water.	(-inf.0 , inf.0)
minDrag	The minimum drag acting on the vehicle at all times. At present, this field is only used by FlyingVehicleData and helps determine it's maxSpeed and movement force.	(-inf.0 , inf.0)
minImpactSpeed	Minimum speed at which the vehicle must be traveling for the OnImpact script function to be called.	(-inf.0 , inf.0)
minJetEnergy	Minimum energy required in order to use the vehicle's jet, if it has one.	(-inf.0 , inf.0)
minRollSpeed	*Note: this field currently has no tangible effect in the engine's simulation.*	--
numDmgEmitterAreas	The number of areas on the vehicle that can display damage effects.	(-inf.0 , inf.0)
softImpactSound	The AudioProfile used to produce sounds for soft impacts.	see type
softImpactSpeed	Minimum speed at which the vehicle must be traveling for the soft impact sound to be played.	(-inf.0 , inf.0)
softSplashSoundVelocity	The minimum velocity at which the soft splash sound will be played when impacting water.	(-inf.0 , inf.0)
splashEmitter[0] splashEmitter[1]	Array of pointers to ParticleEmitterData datablocks which will generate splash effects.	see type
splashFreqMod	The simulated frequency modulation of a splash generated by this vehicle. Multiplied along with vehicle speed and time elapsed when determining splash emission rate.	(-inf.0 , inf.0)
splashVelEpsilon	The threshold speed at which we consider the vehicle's movement to have stopped when updating splash effects.	(-inf.0 , inf.0)
triggerDustHeight	Maximum height from the ground at which the vehicle will generate dust.	(-inf.0 , inf.0)
waterWakeSound	The AudioProfile will be used to produce sounds when a water wake is displayed.	see type

A.2.80. WaterBlock

Fields

Field Name	Description	Sample or Range
density	The default water density is one. Meanwhile, the default character density is 10. This means the character will sink upon entering the water. Therefore, if you want the character to be more buoyant, you can adjust either or both parameters.	[0.0 , inf.0)
DepthGradient	Controls the slope between *MinAlpha* and *MaxAlpha*. In older versions of the engine, Melvin implemented this as a sigmoid function, but since version 1.2, it has been implemented using the (more involved) gamma-correction function.	[0.0 , inf.0)
DistortGridScale	This allows you to adjust distortion such that the effect is the same between a large water block and a small water block. There are not set rules really. You'll just have to experiment.	[0.0 , inf.0)
DistortMag	If this vale is not zero, distortion is enabled. Generally, the magnitude of this value should be less than one or the distortion behaves…strangely. Both positive and negative values are legal.	[0.0 , inf.0)
DistortTime	As you might guess, this period of the distort function. It is inversely proportional to the distortion's rate of change. In other words, larger values mean slower distortions and smaller values mean faster distortions. A value of zero (0) is illegal and will cause the texture rendering to fail gracefully.	[0.0 , inf.0)
envMapIntensity	description	[0.0 , inf.0)
envMapOverTexture	If environmental mapping (see .Reflections and Specular Masks' below) is enabled, this texture is rendered when looking down onto the water from above. This represents an environmental reflection on the water's surface.	~/path/filename.png
envMapUnderTexture	As with *envMapOverTexture*, this represents an environmental reflection, but this is the texture you will see if looking up from beneath the water.	~/path/filename.png
FlowAngle	This parameter (in degrees) determines the direction of the translation.	[0.0 , 360.0)
FlowRate	If this value is non-zero, water flow will be enabled. The higher the value, the more quickly textures will translate.	[0.0 , inf.0)
liquidType	These are leftover values from Tribes 2. You may use them for your own purposes.	Water, OceanWater, RiverWater, StagnantWater, Lava, HotLava, CrustyLava, Quicksand
MaxAlpha	As might be intuited, this parameter determine the maximum alpha to use while rendering shoreTexture. This directly affects the multi-texturing equation involving the *surfaceTexture* and *shoreTexture*.	[0.0 , 1.0]
MinAlpha	As might be intuited, this parameter determine the minimum alpha to use while rendering shoreTexture. This directly affects the multi-texturing equation involving the *surfaceTexture* and *shoreTexture*.	[0.0 , 1.0]
position	The object's placement position, not necessarily the current position.	"1.0 2.0 3.0"
removeWetEdges	Setting this value true, tells the engine to (attempt to) clip the edges of water that protrude from beneath terrain features. Results will vary when using this feature.	[false , true]
rotation	The object's placement rotation, not necessarily the current rotation.	"1.0 2.0 3.0"
scale	The object's placement scale, not necessarily the current scale.	"1.0 2.0 3.0"

Field Name	Description	Sample or Range
ShoreDepth	Shore rendering is determined by a ray-cast at distinct points across the surface of the water block. The result of this ray-cast returns the distance between the top of the water and the terrain directly below that point on the surface. If this value is greater than or equal to *ShoreDepth,* the engine is instructed to render the *shoreTexture*. If you choose to set this value to zero, the *shoreTexture* will not render at all.	[0.0 , inf.0)
ShoreTexture	We'll talk more about shorelines in a moment, but Torque has the ability to render shorelines differently. When it renders the shoreline, it blends this texture with *surfaceTexture*, giving a nice visual effect.	~/path/filename.png
specularColor	This can be used to change both the color of the resultant highlight and its intensity. This parameter takes a 4-tuple floating-point vector "r g b a".	"1.0 0.5 0.5"
specularMaskTex	This texture is used to make the surface of the water look as if it is reflecting light. Again, this should be some kind of caustic grayscale. The engine does take into account the position and elevation of the sun when rendering the specular effect.	~/path/filename.png
specularPower	This determines how large an area is shiny. Lower values cause more of the specular map to be rendered, versus larger values that will tend to show just a spot of highlighting.	[0 , inf)
submergeTexture[0] submergeTexture[1]	These two textures are only used when *liquidType* is one of the Lava types (Lava, HotLava, or CrustyLava). These two textures are rendered perpendicular to the viewing plane. Additionally they are animated. A suggestion I was given, which I'll pass along, is to use two high quality (say 512x512 instead of the normal 256x256) grayscale caustics for these.	~/path/filename.png
surfaceOpacity	This affects how opaque the combination of *surfaceTexture* and *shoreTexture* is. That is it. A value of zero is not transparent, just very translucent. A value of one is quite opaque. You'll have to adjust this meet your needs.	[0.0 , 1.0]
SurfaceParallax	When *FlowRate* is non-zero, the flow-rate of the oriented *surfaceTexture* is controlled by this value as follows: > 1 - Non-oriented surfaceTexture flows more slowly than oriented surfaceTexture. 1 - Non-oriented surfaceTexture and oriented surfaceTexture flow at same rate. < 1 - Oriented surfaceTexture flows more slowly than non-oriented surfaceTexture. Oriented surfaceTexture counter-flows. 0 - Oriented surfaceTexture remains stationary.	[0.0 , inf.0)
surfaceTexture	This texture is used to define the base water layer(s). This texture is rendered in two layers, with one layer re-oriented at a 45-degree angle (about Z of course). This makes the water more interesting.	~/path/filename.png
TessShore	Controls shore 'detail level'.	[0 , inf)
TessSurface	Controls surface 'detail level'	[0 , inf)
tile	Enable/disable tiling.	[false , true]
UseDepthMask	If this value is false, only the *envMapOverTexture* will be rendered on the top of the water. All other 'surface' textures will be disabled.	[false , true]

Field Name	Description	Sample or Range
viscosity	In addition to choosing whether a character will float or sink in water, we can indirectly adjust how quickly this occurs by changing the *viscosity* of the water. A thicker fluid like, say honey, has a high viscosity, whereas plain water will have a low viscosity. By increasing this value, you create an effect where the player will require more time to float or sink.	(-inf.0 , inf.0)
waveMagnitude	description	(-inf.0 , inf.0)

A.2.81. WheeledVehicle

Class used to represent wheeled vehicles.

Console Methods

getwheelCount	setWheelPowered	setWheelSpring	setWheelSteering
setWheelTire			

getWheelCount()

Purpose
Use the ***getWheelCount*** method to determine how many wheels (hubs) this wheeledVehicle has in it's mesh.

Returns
Returns a value between 0 and 8, specifying the number of hubs the mesh has of the name: hub0, hub1, ... hub7.

Notes
When creating a wheeledVehicle mesh, always create hubs using the name hubX, where X is a value between 0 and 7. Furthermore, always create these hubs in order and in pairs (opposite each other).

For example, If you are creating a four-wheeled vehicle, the hubs should be:

```
   hub0 - front left tire
   hub1 - front right tire
   hub2 - rear left tire
   hub3 - rear right tire
```

See Also
setWheelPowered, setWheelSpring, setWheelSteering, setWheelTire

setWheelPowered(*wheelNum* , *isPowered*)

Purpose
Use the **setWheelPowered** method to set the powered state for the specified wheel (hub).

Syntax
wheelNum – A value between 0 and 7, specifying a specific wheel/hub.
isPowered – A boolean value. If set to true, this wheel will contribute to the powered motion of the vehicle, otherwise it will only affect non-powered motion such as rolling and steering.

Returns
Returns true if ***wheelNum*** was successfully set to powered or un-powered. Will return false if ***wheelNum*** does not specify a valid hub.

Notes
By default, all wheels/hubs are powered.

See Also
getWheelCount

setWheelSpring(*wheelNum* , *springDB*)

Purpose
Use the ***setWheelSpring*** method to assign a WheeledVehicleSpring datablock to the specified hub.

Syntax
wheelNum – A value between 0 and 7, specifying a specific wheel/hub.
springDB – The name of ID of a previously specified WheeledVehicleSpring datablock.

Returns
Returns true if ***wheelNum*** was successfully assigned a new spring datablock, otherwise returns false.

Notes
You may change the springs on a wheeled tire at any time, allowing you to enhance vehicles or make them subject to damage.

Each tire can have its own customized spring datablock.

See Also
getWheelCount

setWheelSteering(*wheelNum*, *steerAngle*)

Purpose
Use the **setWheelSteering** method to set that maximum steering angle (away from center) as ***steerAngle*** (radians) for wheel ***wheelNum***.

This limits the turning angle for the vehicle.

Syntax
wheelNum – A value between 0 and 7, specifying a specific wheel/hub.
steerAngle – An angular value between 0.0 and 1.57 radians specifying maximum turning angle this wheel can take.

Returns
Returns true if ***wheelNum*** was successfully set the requested ***steerAngle***, otherwise returns false.

See Also
getwheelCount

setWheelTire(wheelNum , tireDB)

Purpose
Use the **setWheelTire** method to assign a previously specified WheeledVehicleTire datablock to the requested hub.

Syntax
wheelNum – A value between 0 and 7, specifying a specific wheel/hub.
tireDB – The name of ID of a previously specified WheeledVehicleTire datablock.

Returns
Returns true if ***wheelNum*** was successfully assigned a new WheeledVehicleTire datablock.

Notes
Tires may be changed at any time during the game.

See Also
getWheelCount

A.2.82. WheeledVehicleData

Datablock associated with wheeledvehicle object class.

Fields

Field Name	Description	Sample or Range
engineBrake	Floating-point value specifying how much the engine brakes when the engine is not engaged (i.e. when the forward key is pressed).	--
engineSound	AudioProfile specifying sound to play when engine is engaged.	--
engineTorque	Floating-point value specifying the power of the engine.	--
jetSound	AudioProfile specifying sound to play when jets are engaged.	--
maxWheelSpeed	Floating-point value specifying maximum rotational velocity for tires.	--
squealSound	AudioProfile specifying sound to play when tires break friction.	--
tireEmitter	ParticleEmitterData datablock for tire emitters.	--
wheelImpactSound	AudioProfile specifying sound to play when tires impact 'ground'.	--
brakeTorque	Floating-point value specifying the power of the brakes.	--

A.2.83. WheeledVehicleSpring

Datablock used to describe vehicle suspension(s).

Fields

Field Name	Description	Sample or Range
antiSwayForce	Force which acts to dampen lateral sway introduced when wheels opposite each other are extended at different lengths.	[0.0 , inf.0)
damping	Dampening force which counter-acts the spring's force.	[0.0 , inf.0)
force	The force of the spring. Spring forces act straight up and are applied at the spring's root position, which is defined in the vehicle's shape.	[0.0 , inf.0)
length	The length of suspension travel from the root position.	[0.0 , inf.0)

A.2.84. WheeledVehicleTire

Summary of object.

Fields

Field Name	Description	Sample or Range
kineticFriction	Used in the physical simulation to represent the tire's surface friction when it is slipping (has no traction).	[0.0 , inf.0)
lateralDamping	Measures the dampening force applied against lateral forces generated by the tire. See the lateralForce field documentation for more information on vehicle physics as they relate to wheel forces.	[0.0 , inf.0)

Field Name	Description	Sample or Range
lateralForce	Used in the physical simulation to represent the tire's lateral force. Lateral force can in simple terms be considered left/right steering force. WheeledVehicles are acted upon by forces generated by their tires and the lateralForce measures the magnitude of the force exerted on the vehicle when the tires are deformed along the x-axis. With real wheeled vehicles, tires are constantly being deformed and it is the interplay of deformation forces which determines how a vehicle moves. In Torque's simulation of vehicle physics, tire deformation obviously can't be handled with absolute realism, but the interplay of a vehicle's velocity, its engine's torque and braking forces, and its wheels' friction, lateral deformation, lateralDamping, lateralRelaxation, longitudinal deformation, longitudinalDamping, and longitudinalRelaxation forces, along with its wheels' angular velocity are combined to create a robust real-time physical simulation. For this field, the larger the value supplied for the lateralForce, the larger the effect steering moves can have. In Torque tire forces are applied at a vehicle's wheel hubs.	[0.0 , inf.0)
lateralRelaxation	Measures the relaxing force applied against lateral forces generated by the tire. The lateralRelaxation force measures how strongly the tire effectively un-deforms. See the lateralForce field documentation for more information on vehicle physics as they relate to wheel forces.	[0.0 , inf.0)
logitudinalRelaxation (yes, an engine **typo**)	Measures the relaxing force applied against longitudinal forces generated by the tire. The longitudinalRelaxation force measures how strongly the tire effectively un-deforms. See the longitudinalForce field documentation for more information on longitudinal tire forces, and the laterForce field documentation for more information on general vehicle physics as they relate to wheel forces.	[0.0 , inf.0)
longitudinalDamping	Measures the dampening force applied against longitudinal forces generated by the tire. See the longitudinalForce field documentation for more information on longitudinal tire forces, and the laterForce field documentation for more information on general vehicle physics as they relate to wheel forces.	[0.0 , inf.0)
longitudinalForce	Used in the physical simulation to represent the tire's longitudinal force. Longitudinal force can in simple terms be considered forward/backward movement force. WheeledVehicles are acted upon by forces generated by their tires and the longitudinalForce measures the magnitude of the force exerted on the vehicle when the tires are deformed along the y-axis. See the lateralForce field documentation for more information on wheeled vehicle physics. For this field, the larger the value supplied for the longitudinalForce, the larger the effect acceleration/deceleration moves can have.	[0.0 , inf.0)
mass	The mass of the entire wheel. Used in the physics simulation, see the documentation for the WheeledVehicleData datablock for more information. The wheel's mass does not need to be specified in script.	[0.0 , inf.0)
radius	The tire's radius. The radius is determined from the bounding box of the shape provided in the shapefile field, and does not need to be specified in script. The tire should be built with it's hub axis along the object's Y-axis.	[0.0 , inf.0)

Field Name	Description	Sample or Range
restitution	*Note: this field currently has no tangible effect in the engine's simulation.*	--
shapeFile	The path and file name of a shape file to be used for the wheel. Must adhere to the semantics associated with the Filename datatype, as defined in the engine.	~/path/filename.dts
staticFriction	Used in the physical simulation to represent the tire's surface friction when it is not slipping (has traction).	[0.0 , inf.0)

A.3 Console Functions Quick Reference

A.3.1. OpenAL

The following functions are for the most part wrappers on OpenAL functions. Many of these functions use enumerated values. The following table includes the string equivalents to the OpenAL enums as well as information on flags associated with them:

ALEnum	OpenAL Enum (C++)	Flags
AL_CONE_INNER_ANGLE	AL_CONE_INNER_ANGLE	(Source|Get|Set|Int)
AL_CONE_OUTER_ANGLE	AL_CONE_OUTER_ANGLE	(Source|Get|Set|Int)
AL_CONE_OUTER_GAIN	AL_CONE_OUTER_GAIN	(Source|Get|Set|Float)
AL_DIRECTION	AL_DIRECTION	(Source|Get|Set|Float3)
AL_EXTENSIONS	AL_EXTENSIONS	(Context|Get)
AL_GAIN_LINEAR	AL_GAIN_LINEAR	(Source|Listener|Get|Set|Float)
AL_GAIN	AL_GAIN	(Source|Listener|Get|Set|Float)
AL_LOOPING	AL_LOOPING	(Source|Get|Set|Int)
AL_MAX_DISTANCE	AL_MAX_DISTANCE	(Source|Get|Set|Float)
AL_ORIENTATION	AL_ORIENTATION	(Listener|Set|Float6)
AL_PITCH	AL_PITCH	(Source|Get|Set|Float)
AL_POSITION	AL_POSITION	(Source|Listener|Get|Set|Float3)
AL_REFERENCE_DISTANCE	AL_REFERENCE_DISTANCE	(Source|Get|Set|Float)
AL_RENDERER	AL_RENDERER	(Context|Get)
AL_VELOCITY	AL_VELOCITY	(Source|Listener|Get|Set|Float3)
AL_VENDOR	AL_VENDOR	(Context|Get)
AL_VERSION	AL_VERSION	(Context|Get)

alxGetChannelVolume(***channelID*** **)**

Purpose
Use the ***alxGetChannelVolume*** function to get the volume setting for a specified channel.

Syntax
channelID – An integer value, equal to or greater than 0, corresponding to a valid audio channel.

Returns
Returns volume [0.0, 1.0] for channel specified by ***channelID.***

See Also
alxSetChannelVolume

alxGetListenerf(ALEnum)
alGetListener3f(ALEnum)
alGetListeneri(ALEnum)

Purpose
Use the ***al*GetListener**** function to get the current value of a listener parameter, as specified by ***ALEnum***.

Syntax
ALEnum – A string containing an OpenAL enumerated type name. See (above) table of ALEnum values for legal values.

Returns
Returns a float (alxGetListenerf), a vector of three floats (alGetListener3f), or an integer value respectively (alGetListeneri).

Notes
Depending on the ALEnum you need to acquire, be sure to use the correct version (i.e. correct return type) of al*GetListener*.

See Also
alxGetSource*

alxGetSourcef(handle , ALEnum)
alxGetSourcei(handle , ALEnum)
alxGetSource3f(handle , ALEnum)

Purpose
Use the ***alxGetSource**** function to get the current value of a source parameter, as specified by ***ALEnum***.

Syntax
handle – The ID (a non-negative integer) corresponding to a previously set up sound source.
ALEnum – A string containing an OpenAL enumerated type name. See (above) table of ALEnum values for legal values.

Returns
Returns current value of parameter specified by ***ALEnum*** for source identified by ***handle***.

Notes
Depending on the ALEnum you need to acquire, be sure to use the correct version (i.e. correct return type) of alxGetSource*.

See Also
alxSource*, al*GetListener*

alxGetStreamDuration(handle)

Purpose
Use the ***alxGetStreamDuration*** function to determine the length of a previously set up sound in seconds.

Syntax
handle **–** The ID (a non-negative integer) corresponding to a previously set up sound source.

Returns
Returns -1 for invalid **handle**, and 0.0 to N.M for valid ***handle*** indicating length of scheduled sound in seconds.

See Also
alxGetStreamPosition

alxGetStreamPosition(handle)

Purpose
Use the ***alxGetStreamPosition*** function to get the current play position for a playing sound. Note, this value is a percentage equivalent to the percent of the sound that as already played.

Syntax
handle **–** The ID (a non-negative integer) corresponding to a previously set up sound source.

Returns
Returns -1 for invalid ***handle***, and 0.0 to 1.0 for valid ***handle*** indicating what percentage of the sound file has been played.

See Also
alxGetStreamDuration

alGetString(*ALEnum*)

Purpose
Use the ***alGetString*** function to get the string equivalent to the specified OpenAL enumerated value.

Syntax
ALEnum – A string containing an OpenAL enumerated type name. See (above) table of ALEnum values for legal values.

Returns
Returns a string corresponding to the passed ***ALEnum***.

alGetWaveLen(fileName)

Purpose
Use the ***alGetWaveLen*** function to get the play-length of a specified sound file in milliseconds.

Syntax
fileName – A full path to legally formatted sound file.

Returns
Returns play-length of the WAV file specified by ***filename*** in milliseconds.

See Also
alxGetStreamDuration, alxGetStreamPosition

alListener3f(ALEnum , x , y , z)
alxListenerf(AlEnum , value)

Purpose
Use the ***al*Listener**** function to set a listener parameter(s) as specified by the OpenAL enumerated type ***ALEnum***.

Syntax
ALEnum – A string containing an OpenAL enumerated type name. See (above) table of ALEnum values for legal values.
x,y,z – XYZ floating-point coordinates.
value – An ***ALEnum*** type specific value corresponding to the new value for this enumerated parameters.

Returns
No return value.

See Also
al*GetListener*, alxSource*

alxIsPlaying(*handle*)

Purpose
Use the ***alxIsPlaying*** function to determine if the sound associated with a previously set-up sound ***handle*** is playing or not.

Syntax
handle – The ID (a non-negative integer) corresponding to a previously set up sound source.

Returns
Returns 1 if specified ***handle*** is being played, 0 otherwise.

See Also
alxPlay, alxStop, alxStopAll

alxPlay(*handle*)
alxPlay(*profile*)
alxPlay(*profile* , *x* , *y* , *z*)

Purpose
Use the ***alxPlay*** function to start playing a sound specified by either a previously set up sound (handle), or a previously defined audio profile. For 3D sounds, you must specify a XYZ coordinate for the source to play at.

Syntax
handle – The ID (a non-negative integer) corresponding to a previously set up sound source.
profile – The ID (a non-negative integer) corresponding to a previously set up audio profile.
x,y,z – XYZ floating-point coordinates.

Returns
Returns handle to playing sound, or 0 on failure.

See Also
alxIsPlaying, alxStop, alxStopAll

alxSetChannelVolume(*channelD* , *volume*)

Purpose
Use the ***alxSetChannelVolume*** function to set a ***volume*** [0.0, 1.0] for the channel specified by ***channelID***.

Syntax
channelID – An integer value, equal to or greater than 0, corresponding to a valid audio channel.
volume – A value between 0.0 and 1.0 specifying the new volume for the specified channel.

Returns
Returns true on success and false on failure.

See Also
alxGetChannelVolume

alxSourcef(*handle* , *ALEnum* , *value*)
alxSourcei(*handle* , *ALEenum* , *value*)
alxSource3f(*handle* , *ALEnum* , *x* , *y* , *z*)

Purpose
Use the ***alxSource**** function to set a source parameter(s) as specified by the OpenAL enumerated type ***ALEnum***.

Syntax
handle – The ID (a non-negative integer) corresponding to a previously set up sound source.
ALEnum – A string containing an OpenAL enumerated type name. See (above) table of ALEnum values for legal values.
value – An ***ALEnum*** type specific value corresponding to the new value for this enumerated parameters.
x,y,z – XYZ floating-point coordinates.

Returns
No return value.

See Also
alxGetSource*, al*Listener*

alxStop(*handle*)

Purpose
Use the ***alxStop*** function to stop a currently playing sound as specified by ***handle***.

Syntax
handle – The ID (a non-negative integer) corresponding to a previously set up sound source.

Returns
No return value.

See Also
alxIsPlaying, alxPlay, alxStopAll

alxStopAll()

Purpose
Use the ***alxStopAll*** function to stop all currently playing sounds associated with registered handles.

Returns
No return.

See Also
alxIsPlaying, alxPlay, alxStop

OpenALInitDriver()

Purpose
Use the ***OpenALInitDriver*** function to initialize the OpenAL driver.

Returns
Returns true on successful initialization, false otherwise.

Notes
This must be done before <u>all</u> other OpenAL operations.

See Also
OpenALShutdownDriver

OpenALShutdownDriver()

Purpose
Use the ***OpenALShutdownDriver*** function to stop/shut down the OpenAL driver.

Returns
No return value.

Notes
After this is called, you must restart the driver with *OpenALInitDriver* to execute any new sound operations.

See Also
OpenALInitDriver

A.3.2. Debugging

This category of console functions includes functions used to debug or to otherwise examine the scripting environment, the 3D world, engine performance, et cetera.

General

debug()

Purpose
Use the ***debug*** function to cause the engine to issue a debug break and to break into an active debugger.

Returns
No return value.

Notes
For this to work, the engine must have been compiled with either TORQUE_DEBUG, or INTERNAL_RELEASE defined.

dumpConsoleClasses()

Purpose
Use the ***dumpConsoleClasses*** function to prints all registered classes and the console methods associated with them to the console.

Returns
No return value.

Notes
This will dump all classes and methods that were registered from within the engine, AND from the console via scripts.

See Also
dumpConsoleFunctions

dumpConsoleFunctions()

Purpose
Use the ***dumpConsoleFunctions*** function to prints all registered functions to the console.

Returns
No return value.

Notes
This will dump all funtions that were registered from within the engine, AND from the console via scripts.

See Also
dumpConsoleMethods

setEchoFileLoads(enable)

Purpose
Use the ***setEchoFileLoads*** function to enable/disable echoing of file loads (to console).

Syntax
enable – A boolean value. If this value is true, extra information will be dumped to the console when files are loaded.

Notes
This does not completely disable message, but rather adds additional methods when echoing is set to true. File loads will always echo a compile statement if compiling is required, and an exec statement at all times.

Interiors

setInteriorRenderMode(***mode*** **)**

Purpose
Use the ***setInteriorRenderMode*** function to enable various interior rendering modes used for mesh debugging.

Syntax

```
mode - A numeric value between 0 and 16.  Please see the 'Interior Render Modes'
       table below.
```

Returns
```
No return value.
```

Notes
```
For this to work, the engine must have been compiled with TORQUE_DEBUG defined.
```

See Also
```
GLEnableOutline
```

Render Mode	Mode Name	Meaning
0	Normal	Normal.
1	Render Lines	Render interior brush outlines only.
2	Detail	Render interior brushes with flat coloration. White colored blocks indicate brushes that do not change with LOD changes. Red colored blocks will change.
3	Ambiguous	Shows ambiguous polygons. (Good models have none.)
4	Orphan	Shows orphaned polygons. (Good models have none.)
5	Light Map	Shows lightmaps on flat (white) shaded model.
6	Textures Only	Shows textures without lightmaps.
7	Portal Zones	Colorizes portalized zones to make them distinct and easily identifiable.
8	Outside Visible	Marks insides of interiors as white and outsides as red. Tip: An interiors with no portals is marked as all RED.
9	Collision Fans	Displays calculated (by exporter) collision fans with axes showing face directions.
10	Strips	Shows surfaces divided into colorized triangle strips. Each strip has a distinct color from adjacent strips. **Tip:** Large triangles generally give best performance, but strips can be too large in some instance.
11	NULL Surfaces	Renders all faces with NULL texture applied as RED. Tip: Excluding portals, no red surfaces should be visible without taking the camera into walls, or you will have a hole/gap in your surface.
12	Large Textures	All textures large textures will be rendered with a colorized shading: Blue – Width or Height Equal to 256 pixels Green – Width or Height Equal to 512 pixels Red – Width or Height Equal to greater than pixels
13	Hull Surfaces	Renders HULL surfaces with distinct flat colors.
14	Vechile Hull Surfaces	Renders specialized Vechicle HULL surfaces with distinct flat colors.
15	Vertex Colors	-- Currently Unavailable --
16	Detail Levels	Renders entire interior at current LOD coloration. (See LOD colors below).

Level Of Deail (LOD)	Debug Color
0	White
1	Blue
2	Greeen
3	Red
4	Yellow
...	Please see source code.

Journalling

playJournal(*namedFile* , doBreak)

Purpose
Use the ***playJournal*** function to play back a journal from ***namedFile*** and to optionally break (into an active debugger) after loading the Journal. This allow us to debug engine bugs by reproducing them consistently repeatedly.

Syntax
namedFile – A full path to a valid journal file. Usually, journal names end with the extension .jrn.
doBreak – A boolean value. If true, the engine will load the journal and then assert a break (to break into an active debugger). If not true, the engine will play back the journal with no break.

Returns
No return value.

Notes
The journaling system is a vital tool for debugging complex or hard to reproduce engine and script bugs.

See Also
saveJournal

```
playJournal(  "~/myJournal.jrn"  , true );  // Break after loading journal.

playJournal(  "~/myJournal.jrn"  ); // Just play journal
```

saveJournal(namedFile)

Purpose
Use the ***saveJournal*** function to save a new journal of the current game.

Syntax
namedFile – A full path specifying the file to save this journal to. Usually, journal names end with the extension .jrn.

Returns
No return value.

See Also
playJournal

```
saveJournal( "~/myJournal.jrn" );
```

Logging

inputLog(fileName)

Purpose
Use the ***inputLog*** function to enables/disable logging of DirectInput events to a log file specified by ***fileName***.

Syntax
fileName – A valid path to a file in which to store a log of DirectInput events.

Returns
For this to work, the engine must have been compiled with LOG_INPUT defined.

Notes
Once started, input logging cannot be stopped, so only apply this in debug scenarios.

See Also
setLogMode

```
inputLog( "DirectInput.log");
```

setLogMode(mode)

Purpose
Use the ***setLogMode*** function to set the logging level based on bits that are set in the ***mode*** argument.

Syntax
mode – A bitmask enabling various types of logging. See 'Logging Modes' table below.

Returns
No return value.

Notes
This is a general debug method and should be used in all but release cases and perhaps even then.

See Also
intputLog

```
SetLogMode( 6 ); // Dump console output before this to file and leave file open.
```

mode bitmasks	Description
1	Open file and append. Close file on each log write. This allows us to edit the file in a separate editor without having to quit and without getting a file lock conflict.
2	Open file and leave it open. This is more efficient for lots of logging, but we may not be able to view the file till we exit the game. (Note: *NIX users can just tail the file: 'tail -fn 100 filename')
4	Dump anything that has been printed to the console so far. This is needed because the console doesn't get turned on right away, and some output would otherwise be missed.

Memory

dumpUnflaggedAllocs(fileName)

Purpose
Use the ***dumpUnflaggedAllocs*** function to dump all allocations that were made subsequent to a call to *flagCurrentAllocs*. This function, in association with *flagCurrentAllocs*, is used for detecting memory leaks and analyzing memory usage in general.

Syntax
filename – A valid path and filename in which to dump the current memory allocation information.

Returns
No return value.

Notes
For this to work, the engine must have been compiled with DEBUG_GUARD defined.

See Also
flagCurrentAllocs

flagCurrentAllocs()

Purpose
Use the ***flagCurrentAllocs*** function to mark all current memory allocations in preparation for a subsequent call or calls to *dumpUnflaggedAllocs*. This function, in association with *dumpUnflaggedAllocs*, is used for detecting memory leaks and analyzing memory usage in general.

Returns
No return value.

Notes
For this to work, the engine must have been compiled with DEBUG_GUARD defined.

See Also
dumpUnflaggedAllocs

freeMemoryDump()

Purpose
Use the ***freeMemoryDump*** function to dump the current 'memory free' statistics to the console.

Returns
No return value.

Notes
This does not print how much memory is free, but rather an analysis of 'free chunks' of memory.

Metrics

GLEnableLogging(enable)

Purpose
Use the ***GLEnableLogging*** function to enable/disable the gathering of OpenGL metrics.

Syntax
enable - A boolean value. If set to true, the engine will gather various OpenGL metrics and dump them to a file named gl_log.txt. If set to false, logging is stopped, the last writes to the log are flushed, and the file is closed.

Returns
No return value.

Notes
For this to work, the engine must have been compiled with either TORQUE_DEBUG or INTERNAL_RELEASE defined. Always be sure to do a disable after an enable to flush the last log data to the log file.

See Also
GLEnableMetrics, metrics

GLEnableMetrics(enable)

Purpose
Use the ***GLEnableMetrics*** function to enable or disable logging of OpenGL texture and video metrics.

Syntax
enable – A boolean value. When this is set to true, texture and video (triangles and primitives) logging is enabled and dumped as part of calls to certain *metrics*.

Returns
No return value.

Notes
For this to work, the engine must have been compiled with either TORQUE_DEBUG or INTERNAL_RELEASE defined. Use the *metrics* function to get at this information. Also, once this feature is enabled, the following globals will be available for inspection/examination:

```
 OpenGL::triCount0 – Terrain triangles
 OpenGL::triCount1 – DIF triangles
 OpenGL::triCount2 – DTS triangles
 OpenGL::triCount3 – Uncategorized triangles

OpenGL::primCount0 - Terrain primitives
OpenGL::primCount1 – DIF primitives
OpenGL::primCount2 – DTS primitives
OpenGL::primCount3 – Uncategorized primitives
```

See Also
GLEnableLogging, metrics

metrics(metric)

Purpose
Use the ***metrics*** function to enable a display of specified metric information in upper left corner of screen.

Syntax
metric – The class of metrics to display. Please see the 'metric' table below for specific metrics.

Returns
No return value.

Notes
For this to work, the engine must have been compiled with TORQUE_DEBUG defined.

See Also
GLEnableMetrics

metric	Measures
audio	**fps metrics** + OH: Open Handles OLH: Open Looping Handles AS: Active Streams NAS: Null Active Streams LAS: Active Looping Streams LS: Total Looping Streams ILS: Inactive Looping Streams CLS: Culled Looping Streams
debug	**fps metrics** + NTL: Texels Loaded. TRP: Percentage of resident memory (on card) used by textures. NP: Number of primitives being rendered. NT: Number of textures in use. NO: Number of objects being rendered.
interior	**fps metrics** + NTL: Texels Loaded. TRP: Percentage of resident memory (on card) used by textures. INP: Number of primitives being rendered, for interiors only. INT: Number of textures in use, for interiors only. INO: Number of objects being rendered, for interiors only.
fps	FPS: Frames Per Second mspf: Milliseconds per frame
time	**fps metrics +** Sim Time: Sim time to date. Time since engine started. Mod: Ticks since engine started.
terrain	**fps metrics +** L0: Numer of terrain blocks rendering at level zero. FMC: Full mipmap count. DTC: Dynamic texture count. UNU: Unused texture count. STC: Static texture count. DTSU: Dynamic texture space used. STSU: Static texture space used. FRB: Terrain blocks not rendered due to full fogging.
texture **Requires:** GLEnableMetrics(true);	**fps metrics +** NTL: Number of texels loaded. TRP: Percentage of resident memory (on card) used by textures. TCM: Texture cache misses.
video **Requires:** GLEnableMetrics(true);	**fps metrics +** TC: Total triangle count. PC: Total primitive count. T_T: Terrain triangle count. T_P: Terrain primitive count. I_T: Interiors triangle count. I_P: Interiors primitive count. TS_T: Shape (DTS) triangle count. TS_P: Shape (DTS) primitive count. ?_T: Uncategorized triangle count. ?_P: Uncategorized primitive count.
vehicle	**fps metrics +** R: Integration retry count. C: Search count. P: Polygon count for vehicles. V: Vertex count for vehicles.

metric	Measures
water	**fps metrics +** Tri#: Water triangle count. Pnt#: Water point (vertex) count. Hz#: Water haze point count.

Networking

dbgSetParameters (*port* , *password*)

Purpose
Use the ***dbgSetParameters*** function to set the debug connection ***password*** for a specific ***port***.

Syntax
port - The IP port to set the password on.
password - The password for this port. Set this to a NULL string to clear the password for the port.

Returns
No return value.

```
dbgSetParameters( 1130 , “edochi” );
```

dNetSetLogging(enable)

Purpose
Use the ***dNetSetLogging*** function to enable (or disable) network packet logging to the console.

Syntax
enable - A boolean value. If set to true, network packet logging is enabled, otherwise it is disabled.

Returns
No return value.

```
dnetSetLogging(1);
```

Profiling

profilerDump()

Purpose
Use the ***profilerDump*** function to dump engine profile statistics to the console.

Returns
No return value.

Used to dump NetStringTable statistics to the console

```
profilerDump();
```

profilerDumpToFile(filename)

Purpose
Use the ***profilerDumpToFile*** function to dump engine profile statistics to a file.

Syntax
filename – A string value specifying a full or partial path to a file for writing the profiler statistics to.

Returns
No return value.

profilerEnable(enable)

Purpose
Use the ***profileEnable*** function to enable (or disable) engine profiling.

Syntax
enable – A boolean value. If set to true and the engine was compiled with DEBUG specified, engine profiling is enabled, otherwise it is disabled.

Returns
No return value.

Note
TGE has predefined profiling areas surrounded by markers, but you may need to define additional markers (in C++) around areas you wish to profile, by using the PROFILE_START(markerName); and PROFILE_END(); MACROS.

Please read GPGTGE Volume 2 to learn more about this.

```
profilerEnable(false);
```

profilerMarkerEnable(markerName , enable)

Purpose
Use the ***profilerMarkerEnable*** function to enable (or disable) specific profiling markers.

Syntax
markerName – The name of a marker (as specified using the C++ MACRO PROFILER_START) to be enabled/disabled.
enable – A boolean value. If set to true, the specified marker will be enabled for profiling, otherwise it will be disabled.

Returns
No return value.

```
profilerMarkerEnable( mark , true );
```

profilerReset( )

Purpose
Use the ***profilerReset*** function to reset the profile gathering mechanism. This clears the current counters, but all markers retain their current enable/disable status.

Returns
No return value.

Tracing

backtrace()

Purpose
Use the ***backtrace*** function to print the current callstack to the console. This is used to trace functions called from withing functions and can help discover what functions were called (and not yet exited) before the current point in your scripts.

Returns
No return value.

trace(enable)

Purpose
Use the ***trace*** function to enable (or disable) function call tracing. If enabled, tracing will print a message every time a function is entered, showing what arguments it received, and it will print a message every time a function is exited, showing the return value (or last value of last statement) for that function.

Syntax
enable – A boolean value. If set to true, tracing is enabled, otherwise it is disabled.
Returns
No return value.

A.3.3. String Manipulation

Bad Word Filtering

Torque includes a bad word filtering feature that can be used process strings and to replace any words matching words on the list of known bad works. This list is build manually. Bad words are replaced with a user-defined set of random characters. Please note, a small but common list of bad words is already included in the engine. Please refer to the engine source code for this list as a would-be tasteless the list of words here.

addBadWord(aBadWord)

Purpose
Use the ***addBadWord*** function to add new 'bad words' to the current 'bad word' list. Once a word is added to the list, it may not be removed. Also, adding a word more than once has no additional effect.

Syntax
aBadWord – A word to be considered as foul language and to be looked for when the *containsBadWords* and/or *filterString* functions are called.

Returns
No return value.

Notes
Several bad words are already added by the engine. Please refer to the source code for a full listing.

See Also
containsBadWords, filterString

containsBadWords(string)

Purpose
Use the ***containsBadWords*** function to check ***string*** for any previously specified 'bad words'.

Syntax
string – Any string to be checked for bad words.

Returns
Returns true if any words on the bad word list are found in the string. Returns false otherwise.

Notes
This function will catch whole words and variants as long as the variant contains the full spelling of the bad word. For example, if 'wack' where defined as a bad word, this function would return true if the word 'wacko' were included in a sentence.

See Also
addBadWord, filterString

Returns true if ***string*** contains any known bad words.

string filterString(baseString [, replacementChars])

Purpose
Use the ***filterString*** function to parse baseString for words on the 'bad word' list and to then replace those words with random characters.

Syntax
baseString - The original string, possibly containing bad words.
replacementChars - This optional argument can be used to supply a single character or a list of chracters to use for the replacement of characters the bad words.
If no replacement characters are specified, the engine will choose randomly from lower-case alpha-characters (a..z).

Returns
Returns a 'cleaned' string. This string will contain all the original words, except that any words which were deemed 'bad' will have had each character in the word replaced with a random replacement character.

Notes
This function will catch whole words and variants as long as the variant contains the full spelling of the bad word. For example, if 'wack' where defined as a bad word, this function would return true if the word 'wacko' were included in a sentence.

See Also
addBadWord, containsBadWords

Bad Word Filtering Sample

```
function badWordTest()
{
   addBadWord( "poop" );

   %testPhrase = "This is the poop!";

   %filteredPhrase = filterString( %testPhrase , "@#$*" );

   echo("Contains bad words? ==> " , containsBadWords( %testPhrase ) );

   echo("Before filtering: ", %testPhrase );

   echo("After  filtering: ", %filteredPhrase );

}


badWordTest();

Contains bad words? ==> 1
Before filtering: This is the poop!
After  filtering: This is the $@@*!
```

Comparison

strcmp(string1 , string2)

Purpose
Use the ***strcmp*** function to do a lexicographic <u>case sensitive</u> string comparison between ***string1*** and ***string2***.

Syntax
string1 – String to be compared to string2.
string2 – String to be compared to string1.

Returns
Returns a numeric value:

```
- 1 – string1 is less than string2, including case.
  0 – string1 is equal to string2, including case.
  1 – string1 is greater than string2, including case.
```

See Also
see stricmp, strstr

stricmp(string1 , string2)

Purpose
Use the ***stricmp*** function to do a lexicographic <u>case in-sensitive</u> string comparison between ***string1*** and ***string2***.

Syntax
string1 – String to be compared to string2.
string2 – String to be compared to string1.

Returns
Returns a numeric value:

```
- 1 – string1 is less than string2, ignoring case.
  0 – string1 is equal to string2, ignoring case.
  1 – string1 is greater than string2, ignoring case.
```

See Also
see strcmp, strstr

Comparison Sample

```
function strcmptest()
{
   echo("Lexicographic comparisons are not the same as arithmetic comparisons...");

   echo("100 - 10 == 90, but strcmp( \"100\" , \"10\" ) == " ,
        strcmp( "100" , "10" ) );

   echo("\n", "Don't forget about case-sensitivity...");

   echo("strcmp( \"ABC\" , \"abc\" )  == " ,
        strcmp( "ABC" , "abc" ) , "\n\n, but \n" );

   echo("stricmp( \"ABC\" , \"abc\" ) == " ,
        stricmp( "ABC" , "abc" ) );
}


strcmptest();

Lexicographic comparisons are not the same as arithmetic comparisons...
100 - 10 == 90, but strcmp( "100" , "10" ) == 1

Don't forget about case-sensitivity...
strcmp( "ABC" , "abc" )  == -1

, but

stricmp( "ABC" , "abc" ) == 0
```

Conversion

strlwr(sourceString)

Purpose
Use the ***strlwr*** function to convert all alpha characters in ***sourceString*** to lower-case equivalents.

Syntax
sourceString - The string to be modified.

Returns
Returns a copy of ***sourceString*** in which all upper-case characters have been converted to lower-case letters.

See Also
strupr

```
echo( strlwr( "ABCD123" ) ); // Prints abcd123
```

strupr(sourceString)

Purpose
Use the ***strupr*** function to convert all alpha characters in ***sourceString*** to upper-case equivalents.

Syntax
sourceString - The string to be modified.

Returns
Returns a copy of ***sourceString*** in which all lower-case characters have been converted to upper-case letters.

See Also
strlwr

```
echo( strlwr( "abcd123" ) ); // Prints ABCD123
```

Fields (Newline or Tab Separated String)

A field is a sub-string withing a larger string, where each field is delimted by a **NEWLINE** character or a **TAB** character. A newline can be represented as either "**\n**" or the keyword **NL**, and a tab can be represented by hitting the TAB key, or by the keyword **TAB**.

getField(sourceString , index)

Purpose
Use the ***getField*** function to get the field at ***index*** in ***sourceString***.

Syntax
sourceString - A string containing one or more fields.

Returns
Returns field at ***index*** in ***sourceString***, or null string if no field exists at that index.

See Also
getFields, setField

```
// Prints Torque
echo( getField( "GPGTGE + " NL "AND" TAB "Torque" TAB "Rocks" , 2 ) );
```

getFieldCount(sourceString)

Purpose
Use the ***getFieldCount*** function to get the number of fields in ***sourceString***.

Syntax
sourceString – A string containing one or more fields.

Returns
Returns number of fields in ***sourceString*** or 0 if no fields are present.

```
%fields = "You" TAB "must" TAB "buy" TAB "GPGTGE";

echo( getFieldsCount( %fields ) ); // Prints 4
```

getFields(sourceString , index [, endindex])

Purpose
Use the ***getFields*** function to retrieve a set of fields from a ***sourceString***.

Syntax
sourceString – A string containing one or more fields.
index – The index of the first field to retrieve.
endindex – The index of the final field to retrieve.

Returns
Returns all fields (separated by current delimiter) from ***sourceString***, starting at ***index*** and ending at ***endIndex*** or end of string, whichever comes first. If no ***endIndex*** is specified, all remaining fields are returned.

See Also
getField, setField

```
// Prints Torque^Rocks (^ is printed TAB in console)
echo( getFields( "GPGTGE + " NL "AND" TAB "Torque" TAB "Rocks" , 2 , 3 ) );

// Also prints Torque^Rocks
echo( getFields( "GPGTGE + " NL "AND" TAB "Torque" TAB "Rocks" , 2 ) );
```

removeField(sourceString , index)

Purpose
Use the ***removeField*** function to remove a single indexed field from a ***sourceString***.

Syntax
sourceString - A string containing one or more fields.
index - The index of the field to remove.

Returns
Returns ***sourceString*** minus the removed field. If the index is greater than the number of fields in ***sourceString***, the original string is returned.

See Also
setField

```
%fields = "Torque" TAB "So" TAB "Totally" TAB "Rocks!";


// %fields will contain three fields: "Torque", "Totally", and "Rocks!".
%fields = removeField( %fields, 1 );
```

setField(sourceString , index , replace)

Purpose
Use the ***setField*** function to replace an existing field with a new field(s), or to add field(s) to a string..

Syntax
sourceString - A string containing one or more fields.
index - The index of the field to remove.
replace - The new field(s) to replace the field at ***index*** with.

Returns
There are multiple return cases:
- In the first case, a simple one-to-one replacement, the field at ***index*** in ***sourceString*** will be replaced with the value in ***replace***, and the new string will be returned.

- In the first case, a multi-to-one replacement, the field at ***index*** in ***sourceString*** will be replaced with the value in ***replace***, which can be two or more fields, and the new string will be returned.

- In the thrid and final case, new records, empty or filled, can be appended to the end of ***sourceString***. If ***index*** is beyond the end of the ***sourceString***, that is, the ***index*** is greater than the total count of fields in ***sourceString***, the requisite number of empty (null-string) fields will be appended to the end of ***sourceString*** and the value in ***replace*** will be appended to the end of this new string. This entire resultant string will be returned.

See Also
getField, getFields, removeField

```
// %fields will contain two fields, "Torque" and "Rocks!",
%fields =  setField( "Torque" TAB "Rock" , 1 , "Rocks!" );
```

Metrics

strlen(***string*** **)**

Purpose
Use the ***strlen*** function to determine how many characters there are in ***string***.

Syntax
string - The string to count characters for.

Returns
Returns the number of characters in ***string***, or 0 if ***string*** is invalid or a NULL string.

```
// Prints 10
echo( strlen( "0123456789" ) );
```

Records (Newline Separated String)

A record is a sub-string withing a larger string, where each record is delimted by a **NEWLINE** character. A newline can be represented as either "**\n**" or the keyword **NL**.

getRecord(sourceString , index)

Purpose
Use the ***getRecord*** function to get the record at ***index*** in ***sourceString***.

Syntax
sourceString - A string containing one or more records.

Returns
Returns record at ***index*** in ***sourceString***, or NULL string if no record exists at that index.

See Also
getRecords, setRecord

```
// Prints Torque
echo( getRecord( "GPGTGE + " NL "AND" TAB "Torque" TAB "Rocks" , 2 ) );
```

getRecordCount(sourceString)

Purpose
Use the ***getRecordCount*** function to get the number of records in ***sourceString***.

Syntax
sourceString – A string containing one or more records.

Returns
Returns number of records in ***sourceString*** or 0 if no records are present.

```
%records = "You" TAB "must" TAB "buy" TAB "GPGTGE";

echo( getRecordsCount( %records ) ); // Prints 4
```

getRecords(sourceString , index [, endindex])

Purpose
Use the ***getRecords*** function to retrieve a set of records from a ***sourceString***.

Syntax
sourceString – A string containing one or more records.
index – The index of the first record to retrieve.
endindex – The index of the final record to retrieve.

Returns
Returns all records (separated by current delimiter) from ***sourceString***, starting at ***index*** and ending at ***endIndex*** or end of string, whichever comes first. If no ***endIndex*** is specified, all remaining records are returned.

See Also
getRecord, setRecord

```
// Prints Torque^Rocks (^ is printed TAB in console)
echo( getRecords( "GPGTGE + " NL "AND" TAB "Torque" TAB "Rocks" , 2 , 3 ) );

// Also prints Torque^Rocks
echo( getRecords( "GPGTGE + " NL "AND" TAB "Torque" TAB "Rocks" , 2 ) );
```

removeRecord(sourceString , index)

Purpose
Use the ***removeRecord*** function to remove a single indexed record from a ***sourceString***.

Syntax
sourceString - A string containing one or more records.
index - The index of the record to remove.

Returns
Returns ***sourceString*** minus the removed record. If the index is greater than the number of records in ***sourceString***, the original string is returned.

See Also
setRecord

```
%records = "Torque" TAB "So" TAB "Totally" TAB "Rocks!";

// %records will contain three records: "Torque", "Totally", and "Rocks!".
%records = removeRecord( %records, 1 );
```

setRecord(sourceString , index , replace)

Purpose
Use the ***setRecord*** function to replace an existing record with a new record(s), or to add record(s) to a string..

Syntax
sourceString - A string containing one or more records.
index - The index of the record to remove.
replace - The new record(s) to replace the record at ***index*** with.

Returns
There are multiple return cases:
- In the first case, a simple one-to-one replacement, the record at ***index*** in ***sourceString*** will be replaced with the value in ***replace***, and the new string will be returned.

- In the first case, a multi-to-one replacement, the record at ***index*** in ***sourceString*** will be replaced with the value in ***replace***, which can be two or more records, and the new string will be returned.

- In the thrid and final case, new records, empty or filled, can be appended to the end of ***sourceString***. If ***index*** is beyond the end of the ***sourceString***, that is, the ***index*** is greater than the total count of records in ***sourceString***, the requisite number of empty (null-string) records will be appended to the end of ***sourceString*** and the value in ***replace*** will be appended to the end of this new string. This entire resultant string will be returned.

See Also
getRecord, getRecords, removeRecord

```
// %records will contain two records, "Torque" and "Rocks!",
%Records =  setRecord( "Torque" TAB "Rock" , 1 , "Rocks!" );
```

Replacing

strreplace(sourceString , from , to)

Purpose
Use the ***strreplace*** function to replace every instance of ***from*** in ***sourceString*** with ***to***.

Syntax
sourceString - The string to do replacement operations on.
from - The old value to be replaced.
to - The new value to replace old values with.

Returns
Returns a new version of ***sourceString*** in which every instance of the value in ***from*** was replaced with the value in ***to***.

Notes
This function is case-sensitive and only does exact matching.

```
// Prints Torque rocks!
echo( strreplace("Torque is cool!" , "is cool" , "rocks" ) );
```

Searching

getSubStr(sourceString , start , count)

Purpose
Use the ***getSubStr*** function to get a sub-string of ***sourceString***, starting at character index ***start*** and ending at character index ***start + count***, or the end-of-string, which ever comes first.

Syntax
sourceString - The string from which to extract a sub-string.
start - The character index at which the extraction starts.
count - The length of the sub-string to extract.

Returns
Returns a string made up of the character at ***start*** in ***sourceString*** and ending at the end of the original sourceString, or ***start + count***, whichever comes first.

Notes
If ***start + count*** is greater than the length of ***sourceString***, the extraction will return a string shorter than ***count.***

See Also
strchr

```
// Prints 2345
echo( getSubStr( "0123456789" , 2 , 4 ) );
```

strchr(sourceString , char)

Purpose
Use the ***strchr*** function to extract a sub-string of ***sourceString***, where the sub-string is equal to the first occurence of ***char*** in ***sourceString*** followed by the remainder of ***sourceString***.

Syntax
sourceString - The string from which to extract a sub-string.
char - The character to search for in sourceString.

Returns
Returns a string composed of first instance of ***char*** in ***sourceString***, and all of the characters after it. If ***char*** is not found, a NULL string is returned.

See Also
getSubStr

```
// Prints 3456789
echo( strchr( "0123456789" , "3" ) );
```

strpos(sourceString , searchString [, offset])

Purpose
Use the ***strPos*** function to locate the first instance of ***searchString*** in ***sourceString***, starting at character 0, or at an optional ***offset***.

Syntax
sourceString - The string in which to search for ***searchString***.
searchString - The string for which to search for in ***sourceString***.
offset - An optional non-negative integer value representing the character offset within ***sourceString*** at which to begin the search.

Returns
Returns a numeric character index representing the postion in ***sourceString*** at which ***searchString*** was found, or -1 to indicate that no instance of ***searchString*** was found.

See Also
strstr

```
// Prints 2
echo( strpos( "This is a test" , "is" ) );

// Prints 5
echo( strpos( "This is a test" , "is" , 3 ) );
```

strstr(*sourceString* , *searchString*)

Purpose
Use the ***strstr*** function to locate the first instance of ***searchString*** in ***sourceString***.

Syntax
sourceString - The string in which to search for ***searchString***.
searchString - The string for which to search for in ***sourceString***.

Returns
Returns a numeric character index representing the position in ***sourceString*** at which ***searchString*** was found, or -1 to indicate that no instance of ***searchString*** was found.

See Also
strpos

```
// Prints -1
echo( strstr( "This is a test" , "IS" ) );

// Prints 2
echo( strstr( "This is a test" , "is" ) );
```

Stripping and Trimming

stripChars(sourceString , chars)

Purpose
Use the ***stripChars*** function to remove ***chars*** from ***sourceString***.

Syntax
sourceString - The string to be modified.
chars - The character or characters to search for and remove.

Returns
Returns a copy of ***sourceString***, from which all instances of ***chars*** have been removed. This may be the original ***sourceString***, if ***chars*** was not found.

See Also
stripMLControlChars, stripTrailingSpaces

```
// Prints WowThisIsCool
echo( stripChars("WowxThisy*IsCoolz!" , "*xyz" ) );
```

stripMLControlChars(sourceString)

Purpose
Use the ***stripMLControlChars*** function to remove all Torque Markup-Language (ML) symbols from ***sourceString***.

Syntax
sourceString - The string to be modified.

Returns
Returns a copy of ***sourceString*** with all the ML symbols removed, or the original string if no ML symbols were present.

Caution
This may not remove <br> correctly, so check before you trust this function.

See Also
stripChars, stripTrailingSpaces

stripTrailingSpaces(sourceString)

Purpose
Use the ***stripTrailingSpaces*** function to remove all spaces and underscores from ***sourceString***.

Syntax
sourceString - The string to be modified.

Returns
Returns modified string, or original string if no trailing spaces found found.

Notes
This function DOES NOT remove TAB characters, only a space " ", or underline "_" are considered spaces.

See Also
stripChars, stripMLControlChars, ltrim, rtrim, trim

```
// Prints This is a test.
echo( stripTrailingSpaces( "This is a    " ) SPC "test." );
```

ltrim(sourceString)

Purpose
Use the ***ltrim*** function to strip the leading white space from ***sourceString***.

Syntax
sourceString - The string to be trimmed.

Returns
Returns ***sourceString*** with all the leading white spaces removed.

Notes
White space is any character in this set: spaces, TABs, and NULL strings.

See Also
stripChars, stripMLControlChars, stripTrailingSpaces, rtrim, trim

```
// Prints This is a test.
echo( ltrim( " " TAB " " SPC "This is a test." ) );
```

rtrim(sourceString)

Purpose
Use the ***rtrim*** function to strip the trailing white space from ***sourceString***.

Syntax
sourceString - The string to be trimmed.

Returns
Returns ***sourceString*** with all the trailing white spaces removed.

Notes
White space is any character in this set: spaces, TABs, and NULL strings.

See Also
stripChars, stripMLControlChars, stripTrailingSpaces, ltrim, trim

```
// Prints This is a test.
echo( rtrim( "This is a    " ) SPC "test." );
```

trim(sourceString)

Purpose
Use the ***trim*** function to strip the leading and trailing white space from ***sourceString***.

Syntax
sourceString - The string to be trimmed.

Returns
Returns ***sourceString*** with all the leading and trailing white spaces removed.

Notes
White space is any character in this set: spaces, TABs, and NULL strings.

See Also
stripChars, stripMLControlChars, stripTrailingSpaces, ltrim, rtrim

```
// Prints This is a test.
echo( trim( " " TAB " " SPC "This is a     " ) SPC "test." );
```

Tokens

nextToken(tokenList , tokenVar , delimeter)

Purpose
Use the ***nextToken*** function to get the first token found in ***tokenList***, where tokens are separated by the character(s) specified in ***delimeter***. The token itself is stored in a variable whose name is specified in ***tokenVar***. This function provides complex power in a simple package. Please read the notes below, they are very important.

Syntax
tokenList - The string containing token(s).
tokenVar - The 'name' of the variable to store the token in.
delimeter - The character(s) to use as a delimeter. A delimeter may be a single character, or a sequence of characters.

Returns
Returns a copy of tokenList, less the first token and the first delimiter. If there are no more tokens, a NULL string is returned.

Notes
This function is scope-smart. That is, when we specify the name of the variable to store a token in by passing a value in ***tokenVar***, we do not include either a local symbol (%), or a global symbol ($). We just pass in an un-adorned name, let's say "George". Then, depending on where this function is called, "George" will become a local (%George), or a global ($George) variable. If this function is called within a function or method definition, "George" will be local (%George). If this function is called from the file-scope (executed as part of a file and not within the scope of a function or method), "George" will become a global ($George). There is one additional special case. If you attempt to use this from the console command line, the token will vaporize and no variable will be created.

```
function testTokens( )
{

   %myTokens = "A,E,I,O,U";

   while( "" !$= %myTokens )
   {
      %myTokens = nextToken( %myTokens , "theToken" , "," );

      echo( %theToken );

    }
}

testTokens();
A
E
I
O
I
```

Words (Space separated strings)

A word is a sub-string withing a larger string, where each word is delimted by a **SPACE** character. A space can be represented as either " " or the keyword **SPC**.

firstWord(sourceString)

Purpose
Use the ***firstWord*** function to retrieve the first word found in sourceString.

Syntax
sourceString - A string containing one or more words.

Returns
Returns the first word found in ***sourceString***, or a NULL string, if no words are found.

See Also
restWords

```
// a target in range was found so select it
if (%scanTarg)
{
    %targetObject = firstWord(%scanTarg);
    %client.setSelectedObj(%targetObject);
}
```

getWord(sourceString ,index)

Purpose
Use the ***getWord*** function to get the word at ***index*** in ***sourceString***.

Syntax
sourceString – A string containing one or more words.

Returns
Returns word at ***index*** in ***sourceString***, or null string if no word exists at that index.

See Also
getWords, setWord

```
function TerrainEditor::offsetBrush(%this, %x, %y) {
    %curPos = %this.getBrushPos();
    %this.setBrushPos(getWord(%curPos, 0) + %x, getWord(%curPos, 1) + %y);
}
```

getWordCount(sourceString)

Purpose
Use the ***getWordCount*** function to get the number of words in ***sourceString***.

Syntax
sourceString – A string containing one or more words.

Returns
Returns number of words in ***sourceString*** or 0 if no words are present.

```
if ( %pos != -1 ) {
    %wordCount = getWordCount( %action );
    %mods = %wordCount > 1 ? getWords( %action, 0, %wordCount - 2 ) @ " " : "";
    %object = getWord( %action, %wordCount - 1 );
    switch$ ( %object ) {
    case "upov":   %object = "POV1 up";
    case "dpov":   %object = "POV1 down";
    case "lpov":   %object = "POV1 left";
    case "rpov":   %object = "POV1 right";
    case "upov2":  %object = "POV2 up";
    case "dpov2":  %object = "POV2 down";
    case "lpov2":  %object = "POV2 left";
    case "rpov2":  %object = "POV2 right";
    default:       %object = "??";
    }
    return( %mods @ %object );
} else {
    error( "Unsupported Joystick input object passed to getDisplayMapName!" );
}
```

getWords(sourceString ,index [,endindex])

Purpose
Use the ***getWords*** function to retrieve a set of words from a ***sourceString***.

Syntax
sourceString - A string containing one or more words.
index - The index of the first word to retrieve.
endindex - The index of the final word to retrieve.

Returns
Returns all words (separated by current delimiter) from ***sourceString***, starting at ***index*** and ending at ***endIndex*** or end of string, whichever comes first. If no ***endIndex*** is specified, all remaining words are returned.

See Also
getWord, setWord

```
%pos    = getWords(%obj.getTransform(), 0, 2);
```

removeWord(sourceString ,index)

Purpose
Use the ***removeWord*** function to remove a single indexed word from a ***sourceString***.

Syntax
sourceString - A string containing one or more words.
index - The index of the word to remove.

Returns
Returns ***sourceString*** minus the removed word. If the index is greater than the number of words in ***sourceString***, the original string is returned.

See Also
setWord

```
%cam = removeWord(%cam, 0, getWord(%pos, 0));
```

restWords(sourceString)

Purpose
Use the ***restWords*** function to retrieve all words after the first word in ***sourceString***.

Syntax
sourceString – A string containing one or more words.

Returns
Returns a string containing all the words after the first word found in sourceString, or a NULL string if no words remain after the first word (or if no words at all remain).

See Also
firstWord

```
function Heightfield::showTab(%id)
{
    Heightfield::hideTab();
    %data = restWords(Heightfield_operation.getRowTextById(%id));
    %tab  = getField(%data,1);
    echo("Tab data: " @ %data @ " tab: " @ %tab);
    %tab.setVisible(true);
}
```

setWord(sourceString ,index,replace)

Purpose
Use the ***setWord*** function to replace an existing word with a new word(s), or to add word(s) to a string..

Syntax
sourceString – A string containing one or more words.
index – The index of the word to remove.
replace – The new word(s) to replace the word at ***index*** with.

Returns
There are multiple return cases:
- In the first case, a simple one-to-one replacement, the word at ***index*** in ***sourceString*** will be replaced with the value in ***replace***, and the new string will be returned.

- In the first case, a multi-to-one replacement, the word at ***index*** in ***sourceString*** will be replaced with the value in ***replace***, which can be two or more words, and the new string will be returned.

- In the third and final case, new records, empty or filled, can be appended to the end of ***sourceString***. If ***index*** is beyond the end of the ***sourceString***, that is, the ***index*** is greater than the total count of words in ***sourceString***, the requisite number of empty (null-string) words will be appended to the end of ***sourceString*** and the value in ***replace*** will be appended to the end of this new string. This entire resultant string will be returned.

See Also
getWord, getWords, removeWord

```
function WorldEditor::dropCameraToSelection(%this) {
    if(%this.getSelectionSize() == 0) return;

    %pos = %this.getSelectionCentroid();
    %cam = LocalClientConnection.camera.getTransform();
    // set the pnt
    %cam = setWord(%cam, 0, getWord(%pos, 0));
    %cam = setWord(%cam, 1, getWord(%pos, 1));
    %cam = setWord(%cam, 2, getWord(%pos, 2));
    LocalClientConnection.camera.setTransform(%cam);
}
```

A.3.4. NETWORKING

Tags (NetStringTable)

addTaggedString(string)

Purpose
Use the ***addTaggedString*** function to tag a new string and add it to the NetStringTable.

Syntax
string – The string to tagged and placed in the NetStringTable. Tagging ignores case, so tagging the same string (excluding case differences) will be ignored as a duplicated tag.

Returns
Returns a string (containing a numeric value) equivalent to the string ID for the newly tagged string.

buildTaggedString(format , <arg1, ...arg9>)

Purpose
Use the ***buildTaggedString*** function to build a tagged string using the specified ***format***.

Syntax
enable – A boolean value. If set to true, network packet logging is enabled, otherwise it is disabled.

Returns
No return value.

detag(*tagID*)

Purpose
Use the ***detag*** function to convert a tag to a string. This can only be used in the proper context, i.e. to parse values passed to a client command or to a server command. See 'Remote Procedure Call Samples' below.

Syntax
tagID – A numeric tag ID corresponding to a previously tagged string.

Returns
Returns the string associated with the tag ID.

See Also
commandToClient(), commandToServer().

dumpNetStringTable()

Purpose
Use the ***dumpNetStringTable*** function to dump a list of all the currently registered NetStringTable entries, including the times each has been referenced, the total entry count, and the current 'highest' reference string.

Returns
No return value.

Notes
For this to work, the engine must have been compiled with TORQUE_DEBUG defined.

getTag(taggedString)

Purpose
Use the ***getTag*** function to retrieve the tag ID associated with a previously tagged string.

Syntax
taggedString – A previously tagged string.

Returns
Returns the tag ID of the string. If the string was not previously tagged, it gets tagged and the new tag ID is returned.

getTaggedString(tag)

Purpose
Use the ***getTaggedString*** function to convert a tag to a string. This is not the same a detag() which can only be used within the context of a function that receives a tag. This function can be used any time and anywhere to convert a tag to a string.

Syntax
tag – A numeric tag ID.

Returns
Returns the string corresponding to the tag ID.

removeTaggedString(tag)

Purpose
Use the ***removeTaggedSTring*** function to remove a previously tagged string from the NetStringTable.

Syntax
tag - A number tag ID.

Returns
No return value.

Telnet

gotoWebPage(address)

Purpose
Use the ***gotoWebPage*** function to open a browser and go to the specified URL ***address***.

Syntax
address - A complete and valid URL.

Returns
No return value.

setNetPort(port)

Purpose
Use the ***setNetPort*** function to set the netport that the server will listen on.

Syntax
port – A numeric port ID.

Returns
Returns true if the set worked, false otherwise.

telnetSetParameters(port, consolePass, listenPass [, remoteEcho])

Purpose
Use the ***telnetSetParameters*** function to setup and accept telnet requests on a specific ***port***.

Syntax
port – A numeric port ID.
consolePass – Password for read/write access to the console.
listenPass – Password for read access to the console.
remoteEcho – A boolean value to enable echoing to the client, false by default.

Returns
No return value.

Client-Server Communications

commandToClient(client, func [, arg1, ... , argn])

Purpose
Use the ***commandToClient*** function to issue a remote procedure call on a client.

Syntax

client – The numeric ID of a client gameConnection.
func – The suffix of the remote procedure name to be executed on the client.
arg1 .. argn – Optional arguments to be passed to the remote procedure.

Returns
No return value.

Notes
All arguments (excluding ***client***) may be in tagged or non-tagged format. See 'Remote Procedure Call Samples' below.

commandToServer(func [, arg1, ... , argn])

Purpose
Use the ***commandToServer*** function to issue a remote procedure call the server.

Syntax

func – The suffix of the remote procedure name to be executed on the client.
arg1 .. argn – Optional arguments to be passed to the remote procedure.

Returns
No return value.

Notes
All arguments may be in tagged or non-tagged format. See 'Remote Procedure Call Samples' below.

Game Server

allowConnections(enable)

Purpose
Use the ***allowConnections*** to enable (or disable) remote connections to the local game server.

Syntax
enable – A boolean value enabling, or disabling connections to the local server.

Returns
No return value.

cancelServerQuery()

Purpose
Use the ***cancelServerQuery*** function to cancel a previous query*() call.

Returns
No return value.

See Also
queryLANServers(), queryMasterServer(), querySingleServer()

getServerCount()

Purpose
Use the ***getServerCount*** function to determine the number of game servers found on the last queryLANServers() or queryMasterServer() call.

Returns
Returns a numeric value equal to the number of game servers found on the last queryLANServers() or queryMasterServer() call. Returns 0 if the function was not called, or none were found.

Notes
This value is important because it allows us to properly index when calling setServerInfo().

See Also
queryLANServers, queryMasterServer, setServerInfo()

queryLANServers(port , flags , gametype , missiontype , minplayers , maxplayers , maxbots , regionmask , maxping , mincpu , filterflags)

Purpose
Use the ***queryLANServers*** function to establish whether any game servers of the required specification(s) are available on the local area network (LAN).

Syntax

port	– Look for any game servers advertising at this port. Set to 0 if you don't care what port the game server is using.
flags	– Look for any game servers with these special flags set. Set to 0 for no flags.
gametype	– Look for any game servers playing a game type that matches this string. Set to the NULL string to look for any game type.
missiontype	– Look for any game servers playing a mission type that matches this string. Set to the NULL string to look for any mission type.
minplayers	– Look for any game servers with this number of players or more. Set to 0 for no lower limit.
maxplayers	– Look for any game servers with this number of players or fewer. Set to 0 for no upper limit.
maxbots	– Look for any game servers with this number of AI controlled players or fewer. Set to 0 for no limit.
regionmask	– Look for any master servers, on our master server list, in this region. Set to 0 to examine all regions.
maxping	– Look for any game servers with a PING rate equal to or lower than this. Set to 0 for no upper PING limit.
mincpu	– Look for any game servers with a CPU (clock speed) equal or greater than this. Set to 0 for no CPU (clock speed) limit.
filterflags	– Look for any game servers with this game version number or higher. Set to 0 to find all versions.

Returns
No return value.

See Also
getServerCount, queryMasterServer, setServerInfo, stopServerQuery

queryMasterServer(flags , gametype , missiontype , minplayers , maxplayers , maxbots , regionmask , maxping , mincpu , filterflags)

Purpose
Use the ***queryMasterServer*** function to query all master servers in the master server list and to establish if they are aware of any game servers that meet the specified requirements, as established by the arguments passed to this function.

Syntax

flags	– Look for any game servers with these special flags set. Set to 0 for no flags.
gametype	– Look for any game servers playing a game type that matches this string. Set to the NULL string to look for any game type.
missiontype	– Look for any game servers playing a mission type that matches this string. Set to the NULL string to look for any mission type.
minplayers	– Look for any game servers with this number of players or more. Set to 0 for no lower limit.
maxplayers	– Look for any game servers with this number of players or fewer. Set to 0 for no upper limit.

```
    maxbots - Look for any game servers with this number of AI controlled players
              or fewer.  Set to 0 for no limit.
 regionmask - Look for any master servers, on our master server list, in this
              region. Set to 0 to examine all regions.
    maxping - Look for any game servers with a PING rate equal to or lower than
              this.  Set to 0 for no upper PING limit.
     mincpu - Look for any game servers with a CPU (clock speed) equal or greater
              than this. Set to 0 for no CPU (clock speed) limit.
filterflags - Look for any game servers with this game version number or higher.
              Set to 0 to find all versions.
```

Returns
No return value.

Notes
In order for this function to do anything, a list of master servers must have been previously specified. This list may contain one or more server addresses. A call to this function will search all servers in the list. To specify a list, simply create a set of array entries like this:

```
$pref::Master[0] = "2:192.168.123.15:28002";
$pref::Master[1] = "2:192.168.123.2:28002";
...
```

The format of these values is ==> *Region Number : IP Address : Port Number*

These values should be specified in either the client's or the server's preferences file (prefs.cs). You may specifiy it elsewhere, however be sure that it is specified prior to this function being called and before any other functions that rely on it.

See Also
getServerCount, queryLANServers, setServerInfo, startHeartbeat, stopServerQuery

querySingleServer(address [, flags])

Purpose
Use the ***querySingleServer*** function to re-query a previously queried lan server, OR a game server found with *queryLANServers* or with *queryMasterServer* and selected with *setServerInfo*. This will refresh the information stored by TGE about this server. It will not however modify the values of the $ServerInfo::* global variables.

Syntax
address - The IP address and Port to re-query, i.e. "192.168.123.2:28000".
  flags - **No longer used.**

Returns
No return value.

See Also
getServerCount, queryLANServers, queryMasterServer, setServerInfo, stopServerQuery

setServerInfo(index)

Purpose
Use the ***setServerInfo*** function to set the values of the $ServerInfo::* global variables with information for a server found with *queryLANServers* or with *queryMasterServer*.

Syntax
index – The index of the server to get information about.

Returns
Will return true if the information was successfully set, false otherwise.

See Also
getServerCount, queryLANServers, queryMasterServer, querySingleServer

startHeartbeat()

Purpose
Use the ***startHeartbeat*** function to start advertising this game serer to any master servers on the master server list.

Returns
No return value.

Notes
In order for this function to do anything, a list of master servers must have been previously specified. This list may contain one or more server addresses. Once this function is called, the game server will re-advertise itself to all the master servers on its master server lits every two minutes. To specify a list, simply create a set of array entries like this:

```
$pref::Master[0] = "2:192.168.123.15:28002";
$pref::Master[1] = "2:192.168.123.2:28002";
...
```

The format of these values is ==> *Region Number : IP Address : Port Number*

These values should be specified in either the client's or the server's preferences file (prefs.cs). You may specifiy it elsewhere, however be sure that it is specified prior to this function being called and before any other functions that rely on it.

See Also
queryMasterServer, stopHeartbeat

stopHeartbeat()

Purpose
Use the ***startHeartbeat*** function to stop advertising this game serer to any master servers on the master server list.

Returns
No return value.

See Also
queryMasterServer, startHeartbeat

stopServerQuery()

Purpose
Use the ***stopServerQuery*** function to cancel any outstanding server queries.

Returns
No return value.

See Also
queryLANServers, queryMasterServer, querySingleServer

Statistics/Metrics

getMaxFrameAllocation()

Purpose
Use the ***getMaxFrameAllocation*** function to determine the largest amount of memory that has ever been allocated in the process of rendering a single frame.

Returns
Returns a non-negative integer representing the largest number of bytes temporarily allocated during the rendering of a single frame.

Notes
For this to work, the engine must have been compiled with TORQUE_DEBUG defined. During frame rendering, the engine will dynamically allocate and deallocate memory for various tasks. This function allows us to get a peek at the largest amount of allocation that has occured to date. If this number is very large, we may have a bug or be facing a serious performance issue of one form or another. We want this number to be small.

A.3.5. CONSOLE

call(funcName [, args ...])

Purpose
Use the ***call*** function to dynamically build and call a function.

Syntax
funcName - A string containing the unadorned name of a function to be executed.
args ... - Any arguments that should be passed to the function.

Returns
Returns a string containing the results from the function that is built and called.

See Also
eval

```
// Prints Hello World
call( "echo" , "Hello" , " ", "World" );
```

cls( )

Purpose
Use the ***cls*** function to clear the console output.

Returns
No return value.

collapseEscape(text)

Purpose
Use the ***collapseEscape*** function to replace all escape sequences ('\\xx') with a collapsed version ('\xx').

Syntax
text - A string, possibly containing escape sequences.

Returns
Returns a copy of ***text*** with all escape sequences converted to an encoding.

See Also
expandEscape

```
echo( "edo\\t" ); // Prints edo\t

echo( collapseEscape( "edo\\t" ) ); \\ Prints edo^
```

compile(fileName)

Purpose
Use the ***compile*** function to pre-compile a script file without executing the contents.

Syntax
fileName – A path to the script to compile.

Returns
Returns 1 if the script compiled without errors and 0 if the file did not compile correctly or if the path is wrong. Also, ff the path is invalid, an error will print to the console.

See Also
exec

```
compile("./main.cs");
```

deleteVariables(wildCard)

Purpose
Use the ***deleteVariables*** function to delete any global variable matching the ***wildCard*** statement.

Syntax
wildCard – A string identifying what variable(s) to delete. All characters used to create a global are allowed and the special symbol "*", meaning 0 or more instances of any character.

Returns
No return value.

```
$edo = "cool";

echo( $edo ); // Prints cool

deleteVariables( "$ed*" ); // Delete all globals starting with $ed

echo( $edo ); // Prints "" because $edo was deleted
```

echo(text [, ...])

Purpose
Use the ***echo*** function to print messages to the console.

Syntax
text - Any valid text string.
... - Any additional valid text string(s).

Returns
No return value.

See Also
error, warn

```
// Prints This is a test
echo( "This is a test" );
```

enableWinConsole(enable)

Purpose
Use the ***enableWinConsole*** function to tell TGE to create an external console window, either as a separate DOS window or as a new window under OSX/Linux/*NIX.

Syntax
enable – A boolean. If this value is set to true, a new console window will be created.

Returns
No return value.

Notes
Subsequent calls to this function do nothing. Only one external console is allowed.

```
 enableWinConsole( true ); // Open an external console.
```

error(text [, ...])

Purpose
Use the ***error*** function to print error messages to the console. These messages usually print in red.

Syntax
text - Any valid text string.
... - Any additional valid text string(s).

Returns
No return value.

See Also
echo, warn

```
// Prints This is a test
error( "This is a test" );
```

eval(script)

Purpose
Use the ***eval*** function to execute any valid ***script*** statement.

Syntax
script – A string containing a valid ***script*** statement. This may be a single line statement or multiple lines concatenated together with new-line characters.

Returns
Returns the result of executing the ***script*** statement.

Notes
If you choose to eval a multi-line statement, be sure that there are no comments or (\\) comment blocks (**\) embedded in the ***script*** string.

See Also
call

exec(fileName [, nocalls [, journalScript]])

Purpose
Use the ***exec*** function to compile and execute a normal script, or a special journal script.

Syntax
fileName – A string containing a path to the script to be compiled and executed.
nocalls – A boolean value. If this value is set to true, then all function calls encountered while executing the script file will be skipped and not called. This allows us to re-define function definitions found in a script file, without re-executing other worker scripts in the same file.
journalScript – A boolean value. If this value is set tot true, and if a journal is being played, the engine will attempt to read this script from the journal stream. If no journal is playing, this field is ignored.

Returns
Returns true if the file compiled and executed w/o errors, false otherwise.

Notes
If $Pref::ignoreDSOs is set to true, the system will use .cs before a .dso file if both are found.

See Also
compile

```
exec( "./main.cs" );
```

expandEscape(text)

Purpose
Use the ***collapseEscape*** function to replace all escape sequences ('\xx') with an expanded version ('\\xx').

Syntax
text - A string, possibly containing escape sequences.

Returns
Returns a copy of ***text*** with all escape sequences expanded.

See Also
collapseEscape

```
echo( "edo\t" ); // Prints edo^

echo( expandEscape( "edo\t" ) ); \\ Prints edo\t
```

export(wildCard [, fileName [, append]])

Purpose
Use the ***export*** function to save all global variables matching the specified name pattern in ***wildCard*** to a file, either appending to that file or over-writing it.

Syntax
wildCard - A string identifying what variable(s) to export. All characters used to create a global are allowed and the special symbol "*", meaning 0 or more instances of any character.
fileName - A string containing a path to a file in which to save the globals and their definitions.
append - A boolean value. If this value is true, the file will be appended to if it exists, otherwise it will be created/over-written.

Returns
No return value.

```
          echo("Exporting server prefs");
          export("$Pref::Server::*", "./server/prefs.cs", False);
```

quit()

Purpose
Use the ***quit*** function to stop the engine and quit to the command line.

Returns
No return value.

```
quit();
```

warn(text [, ...])

Purpose
Use the ***warn*** function to print warning messages to the console. These messages usually yellow or orange.

Syntax
text – Any valid text string.
... – Any additional valid text string(s).

Returns
No return value.

See Also
warn, error

```
// Prints This is a test
warn( "This is a test" );
```

A.3.6. DEVICE IO

activateDirectInput()

Purpose
Use the ***activateDirectInput*** function to activate polling of direct input devices (keyboard, mouse, joystick, et cetera).

Returns
No return value.

See Also
deactivateDirectInput

```
activateDirectInput();
```

activateKeyboard()

Purpose
Use the ***activateKeyboard*** function to enable directInput polling of the keyboard.

Returns
Returns a true if polling was successfully enabled.

See Also
deactivateKeyboard

```
if(activateKeyboard())
                    echo("Keyboard has been activated");
```

deactivateDirectInput()

Purpose
Use the ***deactivateDirectInput*** function to de-activate polling of direct input devices (keyboard, mouse, joystick, et cetera).

Returns
No return value.

See Also
activateDirectInput

```
deactivateDirectInput();
```

deactivateKeyboard()

Purpose
Use the ***deactivateKeyboard*** function to disable directInput polling of the keyboard.

Returns
No return value.

See Also
activateKeyboard

```
deactivateKeyboard();
```

disableJoystick()

Purpose
Use the ***disableJoystick*** function to disable joystick input.

Returns
No return value.

See Also
enableJoystick, getJoystickAxes, isJoystickEnabled

disableMouse()

Purpose
Use the ***disableMouse*** function to disable mouse input.

Returns
No return value.

See Also
enableMouse

```
disableMouse();
```

echoInputState()

Purpose
Use the ***echoInputState*** function to dump the input state of the mouse, keyboard, and joystick to the console.

Returns
No return value.

See Also
activateDirectInput, deactivateDirectInput, activateKeyboard, deactivateKeyboard, disableJoystick, enableJoystick, enableMouse, disableMouse

```
echoInputState();

DirectInput is enabled but inactive.
- Keyboard is enabled and inactive.
- Mouse is disabled and inactive.
- Joystick is disabled and inactive.
```

enableJoystick()

Purpose
Use the ***enableJoystick*** function to enable joystick input if it is present.

Returns
Will return true if the joystick is present and was successfully enabled, false otherwise.

See Also
disableJoystick, getJoystickAxes, isJoystickDetected

```
enableJoystick();
```

enableMouse()

Purpose
Use the ***enableMouse*** function to enable mouse input.

Returns
Returns true if a mouse is present and it was enabled, false otherwise.

See Also
disableMouse

```
enableMouse();
```

getJoystickAxes(instance)

Purpose
Use the ***getJoystickAxes*** function to get the current axes position (x and y) of any intance of a joystick.

Syntax
instance – A non-negative number value selecting a specific joystick instance attached to this computer.

Returns
Returns a string containing the "x y" position of the joystick.

See Also
disableJoystick, enableJoystick, isJoystickDetected

Used to get the current axes of the joystick pointed to by instance, where instance is a numeric value specifying the joystick number.

```
%joyAxes = getJoystickAxes( 3 );
```

isJoystickDetected()

Purpose
Use the ***isJoystickDetected*** function to determine if one or more joysticks are connected to the system.

Returns
Returns true if one or more joysticks are attached and detected, false otherwise.

Notes
This doesn't tell us how many joysticks there are, just that there are joysticks. It is our job to find out how many and to attach them.

See Also
disableJoystick, enableJoystick, getJoystickAxes

```
if( !isJoystickDetected() ) echo( "No Joystick was detected" );
```

lockMouse(isLocked)

Purpose
Use the ***lockMouse*** function to un/lock the mouse.

Syntax
isLocked - A boolean value

Returns
No return value.

```
            function cursorOff()
{
                        if ( $cursorControlled )
                        lockMouse(true);
                        Canvas.cursorOff();
}
```

A.3.7. FILE I/O

TGE provides a file manager maintains a list of all files in the game directory and all sub-directories. The followings functions are used to access this list. In order manipulate (read/write) the contents of a file, see the console object 'fileObject', listed in the 'ConObjects Quick Reference' or refer to the 'TGE I/O' chapter of EGTGE.

modpath

When a Torque game starts up a call, or calls are made to setModPaths(). This function is passed a game directories, separated by semi-colons (;). The file manager will examine each of these game paths and their sub-directories, building up a list of known files. This information is used later for relative pathing and file searching.

Pathing

Torque supports both direct pathing and relative pathing.

Direct paths start with a slash (/). For example, "/starter.fps/main.cs" points to the file "main.cs" under the game directory "starter.fps", where "starter.fps" is in the root directory (the directory where the executable is started from.

Relative pathing can be accomplished in three basic ways.

In the first method, if an unadorned name is used as the first part of a directory ("starter.fps/test.cs"), the engine will assume that this first name is the name game directory, but then if the unadorned name does not match the game directory, the file match/search will fail. In general, you should not use unadorned names.

In the second method, if a tilde (~) is used as the first part of a directory ("~/test.cs"), the engine will assume that the tilde should be replaced by the root-child this file lives in. For example, the root child of "starter.fps/client/doit.cs" is "starter.fps". Therefore, if we use this path "~/somefile.cs" in a command within the file "doit.cs", the path will be expanded to be "starter.fps/somefile.cs".

In the third and final relative pathing method, if a dot (.) is used as the first part of a directory ("./test.cs"), then the dot is expanded to be the root- path that the current file resides in. For example, the root-path of "starter.fps/client/doit.cs" is "starter.fps/client". Therefore, if we use this path "./somefile.cs" in a command within the file "doit.cs", the path will be expanded to be "starter.fps/client/somefile.cs".

Wildcards

Torque supports a single wildcard, the asterisk (*). This wild card can be interpreted to mean "zero or more instances of any chracter(s)".

expandFilename(*filename*)

Purpose
Use the ***expandFilename*** function to convert a relative path name to a full path name.

Syntax
filename - A string containing the relative or full path and file name of an existing or new file.

Returns
Returns a string containing the expanded path to the specified file.

```
expandFilename("~/data/sound/testing.wav");
```

fileBase(*filename*)

Purpose
Use the ***fileBase*** function to get the name of a file from a relative or full path, not including the file extension.

Syntax
filename - A string containing the relative or full path and file name of an existing or new file.

Returns
Returns an unadorned file name without a path or file extension.

See Also
fileExt, fileName, filePath

```
fileBase(“egt/main.cs”); // will return “main”
```

fileExt(*filename*)

Purpose
Use the ***fileExt*** function to get the extension of a file from a relative or full path, not including the file extension.

Syntax
filename - A string containing the relative or full path and file name of an existing or new file.

Returns
Returns a file extension, including the dot (.).

Notes
If asterisks are present in an extension, as passed in ***filename***, they will not be expanded in the return value.

See Also
fileBase, fileName, filePath
Returns the file extension (suffix) for the file specified by ***filename***.

```
fileExt("script.cs"); // will return ".cs"
```

fileName(*filename*)

Purpose
Use the ***fileName*** function to get the filename and extension of a file from a relative or full path, not including the file extension.

Syntax
filename – A string containing the relative or full path and file name of an existing or new file.

Returns
Returns a string containing the full name of a file less any path before the name.

Notes
If asterisks are present in an file name, as passed in ***filename***, they will not be expanded in the return value.

See Also
fileBase, fileExt, filePath

```
fileName("egt/main.cs"); // will return "main.cs"
```

filePath(*filename*)

Purpose
Use the ***fileBase*** function to get all parts of a path up to, but not including the last slash (/).

Syntax
filename – A string containing the relative or full path and file name of an existing or new file.

Returns
Returns a string containing the relative or full path portion of ***filename***.

Notes
If asterisks are present in any part of the path, as passed in ***filename***, they will not be expanded in the return value.

See Also
fileBase, fileExt, fileName

```
filePath("common/ui/defaultProfiles.cs"); // Will return "common/ui"
```

findFirstFile (*pattern*)

Purpose
Use the ***findFirstFile*** function to find the first file matching ***pattern***.

Syntax
pattern – A full or partial path, followed by a full or partial filename, or any combination of these two elements.

Returns
Returns a full path to the first file name matching ***pattern***. Returns a NULL string, when no matches are found.

Notes
This function will search all directories in the modpath, as created by *setModPaths*. Each time this function is called it will reset an internal variable tracking the current position in the file list. So, multiple routines calling this in an overlapping fashion will clobber each other.

See Also
findNextFile, getFileCount, getModPaths, setModPaths

```
findFirstFile("*.cs");
```

findNextFile (*pattern*)

Purpose
Use the ***findNextFile*** function to find the next file matching ***pattern***.

Syntax
pattern – A full or partial path, followed by a full or partial filename, or any combination of these two elements.

Returns
Returns a full path to the next file name matching ***pattern***. Returns a NULL string, when no matches are found.

Notes
This function will search all directories in the modpath, as created by a call to *setModPaths*. Also, this function requires that ***findNextFile*** be called at least with the same pattern, some time prior to calling this function.

See Also
findFirstFile, getFileCount, getModPaths, setModPaths

```
findNextFile( "*.cs" );
```

getFileCount (*pattern*)

Purpose
Use the ***getFileCount*** function to determine how many files exist in modpaths that match the specified ***pattern***.

Syntax
pattern – A full or partial path, followed by a full or partial filename, or any combination of these two elements.

Returns
Returns a zero if no matches are found, or a positive integer value specifying how many matches there were.

See Also
findFirstFile, findNextFile

```
getFileCount("*.cs");
```

getFileCRC(*filename*)

Purpose
Use the ***getFileCRC*** function to calculate the Cyclic-Redundancy-Check (CRC) value for a file as specified by the partial or full path in ***filename***.

Syntax
filename – A string containing the relative or full path and file name of an existing or new file.

Returns
Returns a non-zero positive integer value corresponding to this file's CRC.

Notes
CRC values are useful for checking to see if two same named files are actually the same. If a client file of the same path and name as a file on the server has a different CRC from the server version, then the files are NOT the same, otherwise they highly likely to be the same. Although it is theoretically possible for two non-matching files to have matching CRCs, the odds are very much against it.

```
getFileCRC("/fps/client/scripts/script/cs");
```

isFile(***filename*** **)**

Purpose
Use the ***isFile*** function to determine whether the value in ***filename*** is in fact an existing file.

Syntax
filename – A string containing the relative or full path and file name of an existing file.
Returns
Returns true if the file exists, false otherwise.

See Also
isWriteableFileName

```
isFile("/fps/client/scripts/script.cs");
```

isWriteableFileName(***filename*** **)**

Purpose
Use the ***isWriteableFileName*** function to determine whether the value in ***filename*** is in fact an existing file and it can be written to.

Syntax
filename – A string containing the relative or full path and file name of an existing file.
Returns
Returns true if the file exists and can be written to, false otherwise.

See Also
isFile

```
isWriteableFileName("/fps/client/scripts/script.cs");
```

A.3.8. PACKAGES

activatePackage(packageName)

Purpose
Use the ***activatePackage*** function to activate a package definition and to re-define all functions named within this package with the definitions provided in the package body.

Syntax
packagename – The name or ID of an existing package.

Returns
No return value.

Notes
This pushes the newly activated package onto the top of the package stack.

See Also
deactivatePackage, isPackage

```
activatePackage(Show);
```

deactivatePackage(packageName)

Purpose
Use the ***deactivatePackage*** function to deactivate a package definition and to pop any definitions from this package off the package stack.

Syntax
packagename – The name or ID of an existing package.

Returns
No return value.

Notes
This also causes any subsequently stacked packages to be popped. i.e. If any packages were activated after the one specified in ***packageName***, they too will be deactivated and popped.

See Also
activatePackage, isPackage

```
deactivePackage(Show);
```

isPackage(packageName)

Purpose
Use the ***isPackage*** function to check if the name or ID specified in ***packageName*** is a valid package.

Syntax
packagename – The name or ID of an existing package.

Returns
Returns true if ***packageName*** is a valid package, false otherwise.

See Also
activatePackage, deactivatePackage

```
isPackage(Show);
```

A.3.9. OBJECTS

An object is in the class instance created by the engine or by scripts and available for access in the console.

isObject(handle)

Purpose
Use the ***isObject*** function to check if the name or ID specified in ***handle*** is a valid object.

Syntax
handle – A name or ID of a possible object.

Returns
Returns true if ***handle*** refers to a valid object, false otherwise.

```
isObject(%player);
```

nameToID(objectName)

Purpose
Use the ***nameToID*** function to convert an object name into an object ID.

Syntax
objectName – A string containing the name of an object.

Returns
Returns a positive non-zero value if the name corresponds to an object, or a -1 if it does not.

Notes
This function is a helper for those odd cases where a string will not covert properly, but generally this can be replaced with a statement like: ("someName")

```
nameToId(%player);
```

strToPlayerName(playerName);

Purpose
Use the ***strToPlayerName*** function to convert a string into a valid player name. It will remove various characters to include: comma (,), dot (.), back-slash (\), and tick (').

Syntax
playerName – A string containing a candidate player name.

Returns
Returns a cleaned version of ***playerName*** that is suitable for networking and use by the engine in various instances.

```
%name = stripTrailingSpaces( strToPlayerName( %name ) );
```

A.3.10. EVENT SCHEDULING

cancel(eventID)

Purpose
Use the ***cancel*** function to cancel a previously scheduled event as specified by ***eventID***.

Syntax
eventID – The numeric ID of a previously scheduled event.

Returns
No return value.

See Also
getEventTimeLeft, getScheduleDuration, getTimeSinceStart, isEventPending, schedule, obj.schedule

getEventTimeLeft(eventID)

Purpose
Use the ***getEventTimeLeft*** function to determine how much time remains until the event specified by ***eventID*** occurs.

Syntax
eventID – The numeric ID of a previously scheduled event.

Returns
Returns a non-zero integer value equal to the milliseconds until the event specified by eventID will occur. However, if eventID is invalid, or the event has passed, this function will return zero.

See Also
cancel, getScheduleDuration, getTimeSinceStart, isEventPending, schedule, obj.schedule

getScheduleDuration (eventID)

Purpose
Use the ***getScheduleDuration*** function to determine how long the event associated with eventID was scheduled for.

Syntax
eventID – The numeric ID of a previously scheduled event.

Returns
Returns a non-zero integer value equal to the milliseconds used in the schedule call that created this event. However, if eventID is invalid, this function will return zero.

See Also
cancel, getEventTimeLeft, getTimeSinceStart, isEventPending, schedule, obj.schedule

Returns the time in milliseconds of the event denoted by ***eventID*** as it was originally scheduled.

Returns 0 if eventID is past or invalid.

getTimeSinceStart(eventID)

Purpose
Use the ***getTimeSinceStart*** function to determine how much time has passed since the event specified by ***eventID*** was scheduled.

Syntax
eventID – The numeric ID of a previously scheduled event.

Returns
Returns a non-zero integer value equal to the milliseconds that have passed since this event was scheduled. However, if eventID is invalid, or the event has passed, this function will return zero.

See Also
cancel, getEventTimeLeft, getScheduleDuration, isEventPending, schedule, obj.schedule

isEventPending(eventID)

Purpose
Use the ***isEventPending*** function to see if the event associated with ***eventID*** is still pending.

Syntax
eventID – The numeric ID of a previously scheduled event.

Returns
Returns true if this event is still outstanding and false if it has passed or eventID is invalid.

Notes
When an event passes, the ***eventID*** is removed from the event queue, becoming invalid, so there is no discnerable difference between a completed event and a bad event ID.

See Also
cancel, getEventTimeLeft, getScheduleDuration, getTimeSinceStart, schedule, obj.schedule

```
$Game::Schedule = schedule($Game::EndGamePause * 1000, 0, "onCyclePauseEnd");

if( isEventPending($Game::Schedule) )  echo(“got a pending event);
```

schedule(t , objID || 0 , functionName, arg0, ... , argN)

Purpose
Use the ***schedule*** function to schedule ***functionName*** to be executed with optional arguments at time ***t*** (specified in milliseconds) in the future. This function may be associated with an object ID or not. If it is associated with an object ID and the object is deleted prior to this event occurring, the event is automatically canceled.

Syntax

t	– The time to wait (in milliseconds) before executing ***functionName***.
objID	– An optional ID to associate this event with.
functionName	– An unadorned (flat) function name.
arg0, ... , argN	– Any number of optional arguments to be passed to ***functionName***.

Returns
Returns a non-zero integer representing the event ID for the scheduled event.

See Also
cancel, getEventTimeLeft, getScheduleDuration, getTimeSinceStart, isEventPending, obj.schedule

```
$Game::Schedule = schedule($Game::EndGamePause * 1000, 0, "onCyclePauseEnd");
```

objID.schedule(t , methodName, arg0, ... , argN)

Purpose
Use the ***objID.schedule*** method to schedule ***methodName*** to be executed with optional arguments at time ***t*** (specified in milliseconds) in the future on the object ***objID***. This event is automatically associated with the object ***objID***, and if that object is deleted prior to this event occuring, the event is automatically cancelled.

Syntax

t	– The time to wait (in milliseconds) before executing ***methodName***.
objID	– An ID to associate this event with.
methodName	– An unadorned (flat) function name.
arg0, ... , argN	– Any number of optional arguments to be passed to ***methodName***.

Returns
Returns a non-zero integer representing the event ID for the scheduled event.

Note
This is one of those rare cases where a method is described in the function appendix, but it really needed to be defined here for clarity.

See Also
cancel, getEventTimeLeft, getScheduleDuration, getTimeSinceStart, isEventPending, schedule

```
$Game::Schedule = schedule($Game::EndGamePause * 1000, 0, "onCyclePauseEnd");
```

A.3.11. DATABLOCKS

deleteDataBlocks()

Purpose
Use the ***deleteDataBlocks*** function to cause a server to delete all datablocks that have thus far been loaded and defined.

Returns
No return value.

```
deleteDataBlocks();
```

A.3.12. VIDEO / TEXTURING

addMaterialMapping(materialName, map0 [, ... , map97])

Purpose
Use the ***addMaterialMapping*** function to create a new material map instance. These maps are used by terrain and interiors for creating proper footstep sounds and dust puffs when an avatar treads upon a terrain block or interior surface using the associated texture.

Syntax
materialName - The unadorned texture name this map is associated with. For example, "sand.jpg" would have a ***materialName*** of "sand".
map0 - At least one (required) map. See map chart below for format of these maps.
... , map97 - Up to 96 additional maps.

Returns
No return value.

Notes
There are fewer mapping types than the maximum number of maps this function will accept.

```
addMaterialMapping( "sand" , "sound: 0" , "color: 0.46 0.36 0.26 0.4 0.0" );
```

Map Type	Format	Purpose
Sound Type	**"sound: #"**	Map this surface to the numeric sound type **#**: 0 – Soft, 1 – Hard, 2 – Metal, 3 – Snow.
Detail Texture	**"detail: texture"**	A texture name to be used for a detail texture on surfaces using this material map.
Environment Map	**"environment: texture reflect"**	This material map has an environment map and it should use **texture** for the mapping and give it a reflective factor of **reflect**, where **reflect** is in the range [0.25, 1.0].
Smoke Puff Color	**"color: R G B StartA EndA"**	When a player treads on this surface and has smoke puffs enabled, the smoke will have the color specified in **R, G,** and **B**, where these values are floating point values in the range [0.0, 1.0]. **StartA** is a floating point value in the range [0.0, 1.0] specifying the puff's starting alpha. **EndA** is of the same format and specifies the puff's ending Alpha.

clearTextureHolds()

Purpose
Use the ***clearTextureHolds*** function to free and release any held textures, returning the size of the held textures free.

Returns
Returns the space freed.

Notes
As long as a texture is not currently in use, it will be released.

See Also
dumpTextureStats, flushTextureCache, purgeResources

```
clearTextureHolds();
```

dumpTextureStats()

Purpose
Use the ***dumpTextureStats*** function to dump information about each texture currently in use to the console. This information will be printed in this format:

```
aaa type: (refCount, holding) textureSpace (filename)
```

Output Syntax

```
        type: - Type of this texture. See 'Texture Types' list below.
    refCount - Number of references to this texture.
     holding - Is this texture being held? "yes" or "no"
textureSpace - Bytes used by this texture.
    filename - Full path to this texture.
```

Returns
No return value.

Notes
For this to work, the engine must have been compiled with TORQUE_DEBUG defined.

See Also
clearTextureHolds, flushTextureCache

```
dumpTextureStats();
```

flushTextureCache()

Purpose
Use the ***flushTextureCache*** function to flush the texture cache.

Returns
No return value.

See Also
clearTextureHolds, dumpTextureStats, purgeResources

```
flushTextureCache();
```

getDesktopResolution()

Purpose
Use the ***getDesktopResolution*** function to determine the current resolution of the desktop (not the application).

Returns
Returns a string containing the current desktop resolution, including the width height and the current bits per pixel.

Notes
To get the current resolution of a windowed display of the torque game engine, simply examine the global variable '$pref::Video::resolution'.

See Also
getDisplayDeviceList, getResolutionList, nextResolution, prevResolution, setDisplayDevice, setRes, setScreenMode, switchBitDepth

```
%res = getDesktopResolution():
```

getDisplayDeviceList()

Purpose
Use the ***getDisplayDeviceList*** function to get a list of valid display devices.

Returns
Returns a tab separated list of valid display devices.

See Also
getDesktopResolution, getResolutionList, setRes, setScreenMode, switchBitDepth

```
echo("Display Device(s) :" @ getDisplayDeviceList() );
```

getResolutionList(devicename)

Purpose
Use the ***getResolutionList*** function to get a semicolon separated list of legal resolutions for a specified device.

Syntax
deviceName – A string containing a supported display device.

Returns
Returns a tab separated list of valid display resolutions for ***devicename***.

Notes
Resolutions are always in the form: *width height bpp*, where *width* and *height* are in pixels and *bpp* is bits-per-pixel.

See Also
getDesktopResolution, getDisplayDeviceList, setRes, setScreenMode, switchBitDepth

```
echo("Possible resolutions :” @ getResolutionList( "OpenGL" ) );
```

getVideoDriverInfo()

Purpose
Use the ***getVideoDriverInfo*** function to dump information on the video driver to the console.

Returns
No return value.

```
echo(“Device driver info :” @ getVideroDriverInfo() );
```

isDeviceFullScreenOnly(devicename)

Purpose
Use the ***isDeviceFullScreenOnly*** function to determine if the device specified in devicename is for full screen display only, or whether it supports windowed mode too.

Syntax
deviceName – A string containing a supported display device.

Returns
Returns true if the device can only display full scree, false otherwise.

See Also
getResolutionList

isFullScreen()

Purpose
Use the ***isFullScreen*** function to determine if the current application is displayed in full-screen mode.

Returns
Returns true if the engine is currently displaying full-screen, otherwise returns false.

nextResolution()

Purpose
Use the ***nextResolution*** function to switch to the next valid (higher) resolution for the current display device.

Returns
Returns true if switch was successful, false otherwise.

See Also
getDesktopResolution, prevResolution, getResolutionList, setRes, setScreenMode, switchBitDepth

png2jpg(pngFilename [, quality])

Purpose
Use the ***png2jpg*** function to save a PNG file specified by ***pngFilename*** as a similarly named JPEG file with the optionally specified quality.

Syntax
pngFilename – The path and file name of the PNG file to convert.
quality – An optional quality between 0 and 100. The default quality is 90.

Returns
Returns -1 if the file could not be opened, 0 on other failures, and 1 if the conversion worked.

prevResolution()

Purpose
Use the ***prevResolution*** function to switch to the previous valid (lower) resolution for the current display device.

Returns
Returns true if switch was successful, false otherwise.

See Also
getDesktopResolution, nextResolution, getResolutionList, setRes, setScreenMode, switchBitDepth

screenShot(filename , format)

Purpose
Use the ***screenShot*** function to capture a screen shot and store it in the file specified by filename.

Syntax
filename – Path to file in which to save screenshot.
format – The format to save the file in, PNG or JPG.

Returns
No return value.

See Also
panoramaScreenShot

```
screenshot("capture0.png", "PNG" );
```

setDefaultFov(defaultFOV)

Purpose
Use the ***setDefaultFov*** function to set the default field-of-view (FOV).

Syntax
defaultFOV – A FOV value between 0.0 and 180.0.

Returns
No return value.

See Also
SetFOV

setDisplayDevice(deviceName [, width [, height [, bpp [, fullScreen]]]])

Purpose
Use the ***setDisplayDevice*** function to select a display device and to set the initial ***width***, ***height*** and bits-per-pixel (***bpp***) setting, as well as whether the application is windowed or in ***fullScreen***.

Syntax
deviceName - A supported display device name.
width - Resolution width in pixels.
height - Resolution height in pixels.
bpp - Pixel resolution in bits-per-pixel (16 or 32).
fullScreen - A boolean value. If set to true, the application displays in full-screen mode, otherwise it will attempt to display in windowed mode.

Returns
Returns true on success, false otherwise.

Notes
If no resolution information is specified, the first legal resolution on this device's resolution list will be used. Furthermore, for each optional argument if the subsequent arguments are not specified, the first matching case will be used. Lastly, if the application is not told to display in full screen, but the device only supports windowed, the application will be forced into windowed mode.

See Also
getDesktopResolution, getDisplayDeviceList, getResolutionList, nextResolution, prevResolution, setRes, setScreenMode, switchBitDepth

setFov(FOV)

Purpose
Use the ***setFov*** function to set the current field-of-view (FOV).

Syntax
FOV - A FOV value between 0.0 and 180.0.

Returns
No return value.

See Also
setDefaultFov

setOpenGLAnisotropy(0.0 .. max.f)

Purpose
Use the ***setOpenGLAnisotropy*** function to enable or disable anisotropic filtering.

Syntax
0.0 .. max.f - A value between 0.0 and the maximum anisotropic level supported by the current machine. Selecting a value higher than ***max.f*** results in ***max.f***.

Returns
No return value.

Notes
Anisotropic filtering is somewhat 'expensive' filtering technique that uses more texels than your average filtering technique (bilinear or trilinear) for determining the color of a pixel for cases where more than one texel may be responsible for that pixels color.

setOpenGLInteriorMipReduction(reductionVal)

Purpose
Use the ***setOpenGLInteriorMipReduction*** function to set the texture quality for interiors.

Syntax
reductionVal – An integer value between 0 and 5, with 0 being the lowest quality and 5 being the highest quality.

Returns
No return value.

See Also
setOpenGLMipReduction ,setOpenGLSkyMipReduction

setOpenGLMipReduction(reductionVal)

Purpose
Use the ***setOpenGLMipReduction*** function to control shape texture detail

Syntax
reductionVal – An integer value between 0 and 5, with 0 being the lowest quality and 5 being the highest quality.

Returns
No return value.

See Also
setOpenGLInteriorMipReduction, setOpenGLSkyMipReduction

setOpenGLSkyMipReduction(reductionVal)

Purpose
Use the ***setOpenGLSkyMipReduction*** function to control texture detail for the skybox and clouds.

Syntax
reductionVal – An integer value between 0 and 5, with 0 being the lowest quality and 5 being the highest quality.

Returns
No return value.

See Also
setOpenGLInteriorMipReduction, setOpenGLMipReduction

setOpenGLTextureCompressionHint (hint)

Purpose
Use the ***setOpenGLTextureCompressionHint*** function to select the OpenGL texture compression method.

Syntax
hint – "GL_DONT_CARE", "GL_FASTEST", or "GL_NICEST". (Please refer to an OpenGL text for information on what these mean.)

Returns
No return value.

```
setOpenGLTextureCompressionHint(GL_NICEST);
```

setRes(width , height , bpp)

Purpose
Use the ***setRes*** function to set the screen to the specified ***width***, ***height***, and bits-per-pixel (***bpp***).

Syntax
width – Resolution width in pixels.
height – Resolution height in pixels.
bpp – Pixel resolution in bits-per-pixel (16 or 32).

Returns
Returns true if successful, otherwise false.

See Also
getDesktopResolution, getDisplayDeviceList, getResolutionList, nextResolution, prevResolution, setDisplayDevice, setScreenMode, switchBitDepth

setScreenMode(width , height , bpp , fullScreen)

Purpose
Use the ***setScreenMode*** function to set the screen to the specified ***width***, ***height***, and bits-per-pixel (***bpp***). Additionally, if ***fullScreen*** is set to true the engine will attempt to display the application in full-screen mode, otherwise it will attempt to used windowed mode.

Syntax
width - Resolution width in pixels.
height - Resolution height in pixels.
bpp - Pixel resolution in bits-per-pixel (16 or 32).
fullScreen - A boolean value. If set to true, the application displays in full-screen mode, otherwise it will attempt to display in windowed mode.

Returns
Returns true if successful, otherwise false.

See Also
getDesktopResolution, getDisplayDeviceList, getResolutionList, nextResolution, prevResolution, setDisplayDevice, setRes, switchBitDepth

```
setScreenMode( 1024 , 768 , 32 , false ); // 1024x768 32bpp windowed mode
```

setVerticalSync(enable)

Purpose
Use the ***setVerticalSync*** function to force the framerate to sync up with the vertical refresh rate.

Syntax
enable - A boolean value. If set to true, the engine will only swap front and back buffers on or before a vertical refresh pass.

Returns
Returns true on success, false otherwise.

Notes
This is used to reduce excessive swapping/rendering. There is generally no purpose in rendering any faster than the monitor will support. Those extra 'ergs' can be used for something else.

switchBitDepth()

Purpose
Use the ***switchBitDepth*** function to toggle the bits-per-pixel (bpp) pixel resolution between 16 and 32.

Returns
Returns true on success, false otherwise.

See Also
getDesktopResolution, getDisplayDeviceList, getResolutionList, nextResolution, prevResolution, setDisplayDevice, setRes,

toggleFullScreen()

Purpose
Use the ***toggleFullScreen*** function to switch from full-screen mode to windowed, or vice versa.

Returns
Returns true on success, false otherwise.

videoSetGammaCorrection(gamma)

Purpose
Use the ***videoSetGammaCorrection*** function to adjust the gamma for the video card.

Syntax
gamma – A floating-point value between 0.0 and 1.0.

Returns
No return value.

Notes
The card will revert to it's default gamma setting as long as the application closes normally.

```
videoSetGammaCorrection($pref::OpenGL::gammaCorrection);
```

Texture Types

Type	Description
0	Regular bitmap
1	Same as 1, but the data will be kept after creation
2	Same as 1 except data will not be loaded to OpenGL and cannot be "bound"
3	Internal ONLY.
4	Same as 1, but has "small textures"
5	Terrain texture.
6	Sky Texture
7	Interior Texture
8	Bump Texture
9	Inverted Bump Texture
10	Detail Texture
11	Zero Border Texture

A.3.13. SPECIAL

calcExplosionCoverage(source, targetObject , coverageMask)

Purpose
Use the ***calcExplosionCoverage*** function to calculate the total exposure for ***targetObject*** to an explosion located at ***source*** position and blocked by all objects having a tight as specified by coverageMask. In other words, this function will calculate how much of an explosive force is applied to an object if intervening objects block portions of the explosion.

Syntax
source – A position vector.
targetObject – The name of ID of the object to check coverage for.
coverageMask – A bitmask containing object type masks for all objects that can block this explosion. 0 For none, and -1 for all. For specific types, please see the "Object Types Table" below.

Returns
Returns a value between 0.0 and 1.0, where 0.0 is no coverage and therefore no damage, and when 1.0 is full coverage.

```
%coverage = calcExplosionCoverage(%position, %targetObject,
            $TypeMasks::InteriorObjectType |  $TypeMasks::TerrainObjectType |
            $TypeMasks::ForceFieldObjectType | $TypeMasks::VehicleObjectType);

if (%coverage == 0) continue;
```

$TypeMasks::StaticObjectType	**$TypeMasks::EnvironmentObjectType**
$TypeMasks::TerrainObjectType	**$TypeMasks::InteriorObjectType**
$TypeMasks::WaterObjectType	**$TypeMasks::TriggerObjectType**
$TypeMasks::MarkerObjectType	**$TypeMasks::GameBaseObjectType**
$TypeMasks::ShapeBaseObjectType	**$TypeMasks::CameraObjectType**
$TypeMasks::StaticShapeObjectType	**$TypeMasks::PlayerObjectType**
$TypeMasks::ItemObjectType	**$TypeMasks::VehicleObjectType**
$TypeMasks::VehicleBlockerObjectType	**$TypeMasks::ProjectileObjectType**
$TypeMasks::ExplosionObjectType	

Object Types Table

getControlObjectAltitude()

Purpose
Use the ***getControlObjectAltitude*** function to determine how high above the terrain the control object.

Returns
Returns a floating-point value equal to the distance above the terrain for the current control object. If the object is below the terrain, a zero is returned.

See Also
getControlObjectSpeed, getTerrainHeight

```
getControlObjectAltitude();
```

getControlObjectSpeed()

Purpose
Use the ***getControlObjectSpeed*** function to determine how fast the control object is currently moving.

Returns
Returns a floating-point value equal to the magnitude of the current control objects velocity in the game world in meters-per-second.

See Also
getControlObjectAltitude, getTerrainHeight

```
%player.getControlObjectSpeed();
```

getClipboard()

Purpose
Use the ***getClipboard*** function to get the contents of the GUI clipboard.

Returns
Returns a string equal to the current contents of the copy the clipboard, or a NULL strain if the copy clipboard is empty.

See Also
setClipboard

getModPaths()

Purpose
Use the ***getModPaths*** function to get the current mod path information.

Returns
Returns a string equivalent to the complete current mod path, that is all pads that are visible to the file manager.

See Also
setModPaths

getTerrainHeight(position)

Purpose
Use the ***getTerrainHeight*** function to determine the height of the terrain at an XY ***position*** in the game world.

Syntax
position – An XY position vector in the game world.

Returns
Returns the terrain height at XY position.

See Also
getControlObjectAltitude, getControlObjectSpeed

```
%TerHeight = getTerrainHeight(%pos);
```

isPointInside(position)

Purpose
Use the ***isPointInside*** function to determine if an XYZ point is in side an interior.

Syntax
position – An XYZ position vector in the game world.

Returns
Returns true if the XYZ position is inside an interior, otherwise returns false.

Notes
This will only work if the interior at ***position*** is using portals, otherwise inside and outside are equivalent.

```
isPointInside("143 34  567");
```

pathOnMissionLoadDone()

Purpose
Use the ***pathOnMissionLoadDone*** function to load all path information from the currently loaded interiors.

Returns
No return value.

```
pathOnMissionLoadDone();
```

setModPaths(path)

Purpose
Use the ***setModPaths*** function to set the current mod path to the value specified in ***path***.

Syntax
path – A string containing a semi-colon (;) separated list of game and mod paths.

Returns
No return value.

See Also
getModPaths

```
// Set the mod path which dictates which directories will be visible
// to the scripts and the resource engine.
$modPath = pushback($userMods, $baseMods, ";");
setModPaths($modPath);
```

setClipboard(string)

Purpose
Use the ***setClipboard*** function to Set value on clipboard to ***string***.

Syntax
string – The new value to place in the GUI clipboard.

Returns
Returns true if successful, false otherwise.

See Also
getClipoard

setZoomSpeed(delay)

Purpose
Use the ***setZoomSpeed*** function to set the current zoom speed to the value specified in ***delay***, where ***delay*** is the time it takes to transition across 90-degrees of field-of-view specified in milliseconds. The maximum delay is 2000 ms.

Syntax
delay – An integer value between 0 and 2000, equal to the time it takes the camera across 90-degrees of FOV.

```
setZoomSpeed( $pref::Player::zoomSpeed );
```

A.3.14. RESOURCE MANAGEMENT

dumpResourceStats()

Purpose
Use the ***dumpResourceStats*** function to dump a listing of the currently in-use resources to the console. This will include such things as sound files, font files, etc.

Returns
No return value.

Notes
For this to work, the engine must have been compiled with TORQUE_DEBUG defined.

See Also
purgeResources

```
dumpResourceStats();
```

purgeResources()

Purpose
Use the ***purgeResources*** function to purge all game resources.

Returns
No return value.

See Also
clearTextureHolds, dumpResourceStats, dumpTextureStats, flushTextureCache

```
purgeResources();
```

A.3.15. SCENE

lightScene([completeCallback [, force])

Purpose
Use the ***lightScene*** function to relight a currently open mission.

Syntax
```
completeCallback - A unadorned function name to execute when the re-light is done.
           force - Optionally either "forceAlways" (always relights) , or
                   "forceWritable" (only relight file is writeable).
```

Returns
No return value.

resetLighting()

Purpose
Use the ***resetLighting*** function to reset the OpenGL lighting model.

Returns
No return value.

A.3.16. CONTAINERS and RAYCASTS

ContainerBoxEmpty(mask , location , xRadius [, yRadius , zRadius])

Purpose
Use the ***ContainerBoxEmpty*** function to check for any objects of type mask within a box of variable size located at

Syntax
```
    mask - A bitmask corresponding to the type of objects to check for.  See the
           'Object Types Table' above.
location - An XYZ position vector in the game world, pinpointing the centroid of
           the bounding box.
 xRadius - The radius of the bounding box in the X-axis.
 yRadius - The optional radius of the bounding box in the Y-axis.
 zRadius - The optional radius of the bounding box in the Z-axis.
```

Returns
Returns true, if not objects were found, and false if objects were found.

Notes
If the ***yRadius*** and ***zRadius*** values are not supplied, they are assumed to be equal to ***xRadius***.

See Also
containerFindFirst, containerFindNext, ContainerRayCast

containerFindFirst(mask , location, , xRadius , yRadius , zRadius)

Purpose
Use the ***containerFindFirst*** function to find the first instance of an object matching the type mask within the specified bounding box.

Syntax
```
    mask – A bitmask corresponding to the type of objects to check for.  See the
           'Object Types Table' above.
location – An XYZ position vector in the game world, pinpointing the centroid of
           the bounding box.
 xRadius – The radius of the bounding box in the X-axis.
 yRadius – The optional radius of the bounding box in the Y-axis.
 zRadius – The optional radius of the bounding box in the Z-axis.
```

Returns
Returns 0 if no objects are found, otherwise returns an integer representing the ID of the first object found.

See Also
ContainerBoxEmpty, containerFindNext, ContainerRayCast

containerFindNext()

Purpose
Use the ***containerFindNext*** function to find the next instance of an object in a bounding box search previously initiated with *containerFindFirst*.

Returns
Returns 0 if no objects are found, otherwise returns an integer representing the ID of the next object found.

See Also
ContainerBoxEmpty, containerFindFirst, ContainerRayCast

ContainerRayCast (startPos , endPos , mask [, exempt])

Purpose
Use the ***ContainerRayCast*** function to see if an object matching the specified ***mask*** type is positioned along a ray starting at ***startPos*** and ending at ***endPos***. One object may be marked for expemption from ray cast collisions.

Syntax
startPos – An XYZ vector containing the tail position of the ray.
endPos – An XYZ vector containing the head position of the ray.
mask – A bitmask corresponding to the type of objects to check for. See the 'Object Types Table' above.
exempt – An optional ID for a single object that ignored for this raycast.

Returns
Returns 0 if no objects were struck by the ray, or a non-zero integer representing the ID of the object that was struck.

Notes
The ***exempt*** field is used to keep ray casts originating at an object from hitting the object itself.

See Also
ContainerBoxEmpty, containerFindFirst, containerFindNext

ContainerSearchCurrDist()

Purpose
Use the ***ContainerSearchCurrDist*** function when using *InitContainerRadiusSearch* and *ContainerSearchNext* functions to find objects to determine the distance of the center of the last object found from the center of the current search container.

Returns
Returns a floating point value equal to the distance between center the last found container search item and the center of the container search box.

Notes
Caution! Do not call this function without first setting up a search container or the engine will crash.

See Also
ContainerSearchCurrRadiusDist, ContainerSearchNext, InitContainerRadiusSearch

ContainerSearchCurrRadiusDist()

Purpose
Use the ***ContainerSearchCurrDist*** function when using *InitContainerRadiusSearch* and *ContainerSearchNext* functions to find objects to determine the distance of the closest point of the last object found from the center of the current search container.

Returns
Returns a floating point value equal to the distance between center of the container search box and the point on the last found item that is closest to the container center.

Notes
Caution! Do not call this function without first setting up a search container or the engine will crash.

See Also
ContainerSearchCurrDist, ContainerSearchNext, InitContainerRadiusSearch

```
%rad = ContainerSearchCurrRadiusDist();
```

ContainerSearchNext()

Purpose
Use the ***ContainerSearchNext*** function to find the next object in the currently defined search container.

Returns
Returns non-zero integer value equal to the ID of an object, or zero if no objects were found.

Notes
Caution! Do not call this function without first setting up a search container or the engine will crash.

See Also
ContainerSearchCurrDist, ContainerSearchCurrRadiusDist, InitContainerRadiusSearch

InitContainerRadiusSearch (centerPos , radius , mask)

Purpose
Use the ***InitContainerRadiusSearch*** function to find all objects matching the specified ***mask*** type within a bounding ***radius*** centered at ***centerPos***.

Syntax
centerPos – An XYZ vector specifying the world position of the search container's centroid.
radius – A floating-point value specifying the radius of the search container.
mask – A bitmask corresponding to the type of objects to check for. See the 'Object Types Table' above.

Returns
No return value.

Notes
This search is static. That is it will find all objects within the specified radius and then the found objects can be retrieved with *ContainerSearchNext*. To find new objects, you will have to re-initialize the search.

See Also
ContainerSearchCurrDist, ContainerSearchCurrRadiusDist, ContainerSearchNext

A.3.17. EDITORS

snapToggle()

Purpose
Use the ***snapToggle*** function to enable object placement toggling in the World Editor.

Returns
No return value.

Notes
With toggling enabled, objects will snap to the horizontal world grid when moved with the mouse.

A.3.18. BUILD

getBuildString()

Purpose
Use the ***getBuildString*** function to determine if this build is a "Debug" release, or a "Release" build.

Returns
Returns a string, either "Debug" for a debug build, or "Release" for a release build.

See Also
getCompileTimeString, getVersionNumber, getVersionString, isDebugBuild

getCompileTimeString()

Purpose
Use the ***getCompileTimeString*** function to determine when the currently running engine was built.

Returns
Returns a string containing "*Month Day Year* at *Hour*:*Minute*:*Second*" showing when this executable was built.

See Also
getBuildString, getVersionNumber, getVersionString, isDebugBuild

getVersionNumber()

Purpose
Use the ***getVersionNumber*** function to get the version number of the currently executing engine.

Returns
Returns an integer representing the engine's version number.

See Also
getBuildString, getCompileTimeString, getVersionString, isDebugBuild

getVersionString()

Purpose
Use the ***getVersionString*** function to get the version name and number for the currently executing engine.

Returns
Returns a string containing a name and an integer representing the engine's version type and version number.

See Also
getBuildString, getCompileTimeString, getVersionNumber, isDebugBuild

isDebugBuild()

Purpose
Use the ***isDebugBuild*** function to determine if this is a debug build.

Returns
Returns true if this is a debug build, otherwise false.

See Also
getBuildString, getCompileTimeString, getVersionNumber, getVersionString

A.3.19. TIME

getRealTime()

Purpose
Use the ***getRealTime*** function to the computer time in milliseconds.

Returns
Returns the current real time in milliseconds.

See Also
getSimTime

Used to get the real time (in milliseconds)
Returns a numeric

```
echo(“Time in milliseconds: “ @ getRealTime() );
```

getSimTime()

Purpose
Use the ***getSimTime*** function to get the time, in ticks, that has elapsed since the engine started executing.

Returns
Returns the time in ticks since the engine was started.

See Also
getRealTime

```
function timeMetricsCallback()
{
   return fpsMetricsCallback() @
          "  Time -- " @
          "  Sim Time: " @ getSimTime() @
          "  Mod: " @ getSimTime() % 32;
}
```

A.3.20. GUIS

createCanvas(WindowTitle)

Purpose
Use the ***createCanvas*** function to initialize the canvas.

Returns
Returns true on success, false on failure.

See Also
createEffectCanvas

createEffectCanvas(WindowTitle)

Purpose
Use the ***createEffectCanvas*** function to initialize the effects canvas.

Returns
Returns true on success, false on failure.

See Also
createCanvas

A.3.21. MATH

getBoxCenter(box)

Purpose
Use the ***getBoxCenter*** function to find the centroid of a cube (box).

Syntax
box - A vector containing two three-element floating-point position vectors:
"X1 Y1 Z1 X2 Y2 Z2".

Returns
Returns a vector containing a three-element floating-point position vector equal to the centroid of the area defined by ***box***.

```
%pos = getBoxCenter( %this.getWorldBox() );
```

getRandom( )
getRandom(max)
getRandom(min , max)

Purpose
Use the ***getRandom*** function to get a random floating-point or integer value. This function comes in three forms.

The first *getRandom()* takes no arguments and will return a random floating-point value in the range of 0.0 to 1.0.

The second *getRandom(max)* takes one argument representing the max integer value this will return. It will return an integer value between 0 and ***max***.

The third *getRandom(min , max)* takes two arguments representing the min and max integer values this will return. It will return an integer value between ***min*** and ***max***.

Syntax
min - Minimum inclusive integer value to return.
max - Maximum inclusive integer value to return.

Returns
If no arguments are passed, will return a floating-point value between 0.0 and 1.0. If one argument is passed, will return an integer value between 0 and ***max*** inclusive. If two arguments are passed, will return an integer value between ***min*** and ***max*** inclusive.

Notes
Be sure to recognize the difference between the three variants of *getRandom*. Only the no-args version will return a floating point.

See Also
getRandomSeed

getRandomSeed()

Purpose
Use the ***getRandomSeed*** function to get the current seed for the random generator.

Returns
Returns an integer value representing the current seed of the random generator.

Notes
You can re-seed the generator with this value to allow you to recreate a random sequence. Merely save the seed and execute your random sequence. Later, to reproduce this sequence re-seed the generator with *setRandomSeed* and your saved value. Then, the generator will produce the same random sequence that followed the call to *getRandomSeed*.

See Also
getRandom, setRandomSeed

```
%seed = getRandomSeed();
```

mAbs(val)

Purpose
Use the ***mAbs*** function to get the magnitude of ***val***.

Syntax
val - An integer or a floating-point value.

Returns
Returns the magnitude of ***val***.

```
%abs = mAbs(76.3);
```

mAcos(val)

Purpose
Use the ***mAcos*** function to get the inverse cosine of ***val*** in radians.

Syntax
val - A value between -1.0 and 1.0 equal to the cosine of some angle theta.

Returns
Returns the inverse cosine of ***val*** in radians. This value will be in the range [0 , 3.14159].

See Also
mCos

```
%acos = mAcos(-8,3);
```

mAsin(val)

Purpose
Use the ***mAsin*** function to get the inverse sine of ***val*** in radians.

Syntax
val - A value between -1.0 and 1.0 equal to the sine of some angle theta.

Returns
Returns the inverse sine of ***val*** in radians. This value will be in the range
[- 3.14159/2 , 3.14159/2].

See Also
mSin

mAtan(val)

Purpose
Use the ***mAtan*** function to get the inverse tangent of ***val*** in radians.

Syntax
val - A value between -inf.0 and inf.0 equal to the tangent of some angle theta.

Returns
Returns the inverse tangent of ***val*** in radians. This value will be in the range
[- 3.14159/2 , 3.14159/2].

See Also
mTan

mathInit(extention)

Purpose
Use the ***MathInit*** function to install a specified math extensions, or to detect and enable all extensions.

Syntax
```
extension - Can be any of these:
            detect - Detect all supported extensions and enable.
            C      - Enable standard C extensions.
            FPU    - Enable floating-point-unit extensions.
            MMX    - Enable Intel MMX extensions.
            3DNOW  - Enable AMD 3DNOW extensions.
            SSE    - Enable Intel SSE extensions.
```

Returns
No return value.

Notes
Generally speaking, the best ***extension*** choice is to used detect. This will automatically detected and enable all extensions supported by the current processor. It will also print out a list of the extension that were enabled to the console.

```
mathInit( detect );
```

matrixCreate(posVec , rotVec)

Purpose
Use the ***matrixCreate*** function to create a transform matrix from a three-element floating-point translation vector and a four-element floating-point rotation vector.

Syntax
```
posVec - A three-element floating-point translation vector: "PosX PosY PosZ".
rotVec - A four-element floating-point rotation vector: "RotX RotY RotZ".
         These are rotations about the specified axes.
```

Returns
Returns a transform matrix of the form "PosX PosY PosZ RotX RotY RotZ theta".

See Also
MatrixCreateFromEuler

MatrixCreateFromEuler (rotVec)

Purpose
Use the ***MatrixCreateFromEuler*** function to calculate a transform matrix from a three-element floating-point rotation vector.

Syntax
rotVec - A three-element floating-point rotation vector: "RotX RotY RotZ".
These are rotations about the specified axes.

Returns
Returns a transform matrix of the form "0 0 0 X Y Z theta".

See Also
MatrixCreate

MatrixMulPoint(transform , point)

Purpose
Use the ***MatrixMulPoint*** function to multiply a seven element ***transform*** matrix by a three element ***point*** vector, producing a three element position vector.

Syntax
transform – A seven-element transform matrix.
point – A three-element point/position vector.

Returns
Returns a three-element position vector.

See Also
MatrixMultiply, MatrixMulVector

MatrixMultiply(transformA , transformB)

Purpose
Use the ***MatrixMultiply*** function to multiply two seven-element transform matrices to produce a new seven element matrix.

Syntax
transformA - A seven-element transform matrix of the form "PosX PosY PosZ RotX RotY RotZ theta".
transformB - A seven-element transform matrix of the form "PosX PosY PosZ RotX RotY RotZ theta".

Returns
Returns a seven-element matrix resulting from **transiformA** x ***transformB***.

See Also
MatrixMulPoint, MatrixMulVector

MatrixMulVector(transform , vector)

Purpose
Use the ***MatrixMulVector*** function to multiply a seven-element transform matrix with a three-element matrix.

Syntax
transform - A seven-element transform matrix of the form "PosX PosY PosZ RotX RotY RotZ theta".
vector - A three-element vector.

Returns
Returns three-element resulting from **vector** * ***transform***.

See Also
MatrixMulPoint, MatrixMultiply

mCeil(val)

Purpose
Use the ***mCeil*** function to calculate the next highest integer value from ***val***.

Syntax
val - A floating-point value.

Returns
Returns an integer representing the next highest integer from ***val***.

See Also
mFloor

mCos(val)

Purpose
Use the ***mCos*** function to get the cosine of the radian angle ***val***.

Syntax
val - A value between -3.14159 and 3.14159.

Returns
Returns the cosine of ***val***. This value will be in the range [-1.0 , 1.0].

See Also
mAcos

mDegToRad(val)

Purpose
Use the ***mDegToRad*** function to convert degrees to radians.

Syntax
val - A floating-point number representing some number of degrees.

Returns
Returns the equivalent of the degree value ***val*** in radians.

See Also
mRadToDeg

mFloatLength(val , numDecimals)

Purpose
Use the ***mFloatLength*** function to limit the number of decimal places in ***val*** to ***numDecimals***.

Syntax
```
        val - A floating-point value.
numDecimals - An integer between 0 and inf representing the number of decimal
              places to allow val to have.
```

Returns
Returns a floating-point value equivalent to a truncated version of ***val***, where the new version has ***numDecimals*** decimal places.

mFloor(val)

Purpose
Use the ***mFloor*** function to calculate the next lowest integer value from ***val***.

Syntax
val - A floating-point value.

Returns
Returns an integer representing the next lowest integer from ***val***.

See Also
mCeil

mLog(val)

Purpose
Use the ***mLog*** function to calculate the natural logarithm of ***val***.

Syntax
val - A numeric value.

Returns
Returns the natural logarithm of ***val***.

mPow(val , power)

Purpose
Use the ***mPow*** function to calculated ***val*** raised to the power of ***power***.

Syntax
val - A numeric (integer or floating-point) value to be raised to a power.
piower - A numeric (integer or floating-point) power to raise ***val*** to.

Returns
Returns val^power.

mRadToDeg(val)

Purpose
Use the ***mRadToDeg*** function to convert radians to degrees.

Syntax
val - A floating-point number representing some number of radians.

Returns
Returns the equivalent of the radian value ***val*** in degrees.

See Also
mDegToRad

mSin(val)

Purpose
Use the ***mSin*** function to get the sine of the radian angle ***val***.

Syntax
val – A value between -3.14159 and 3.14159.

Returns
Returns the sine of ***val***. This value will be in the range [-1.0 , 1.0].

See Also
mAsin

mSolveCubic(a , b , c , d)

Purpose
Use the ***mSolveCubic*** function to solve for x0, x1, x2 in third-order polynomial equation with factors a, b, c, d:

aX^3 + bX^2 + cX + d = 0

Syntax
a, b , c, d – Polynomial factors (see above).

Returns
Returns x0, x1, x2: (X + x0) (X + x1) (X + x2)

mSolveQuadratic(a , b , c)

Purpose
Use the ***mSolveCubic*** function to solve for x0, x1 in second-order polynomial equation with factors a, b, c:

aX^2 + bX + c = 0

Syntax
a, b , c – Polynomial factors (see above).

Returns
Returns x0, x1: (X + x0) (X + x1)

Notes
Warning, x0 and x1 are inverted.

mSolveQuartic(a,b,c,d,e)

Purpose
Use the ***mSolveCubic*** function to solve for x0, x1, x2, x3 in fourth-order polynomial equation with factors a, b, c, d, e:

$$aX^4 + bX^3 + cX^2 + dX + e = 0$$

Syntax
a, b , c, d, e- Polynomial factors (see above).

Returns
Returns x0, x1, x2, x3: (X + x0) (X + x1) (X + x2) (X + x3)

Notes
Broken.

mSqrt(val)

Purpose
Use the ***mSqrt*** function to calculated the square root of ***val***.

Syntax
val - A numeric value.

Returns
Returns the the squareroot of ***val***.

mTan(val)

Purpose
Use the ***mTan*** function to get the tangent of the radian angle ***val***.

Syntax
val - A value between -3.14159/2 and 3.14159/2.

Returns
Returns the tangent of ***val***. This value will be in the range [-inf.0 , inf.0].

See Also
mAtan

setRandomSeed(startSeed)

Purpose
Use the ***setRandomSeed*** function to initialize the random number generator with the initial seed of ***startSeed***.

Syntax
startSeed – The new starting seed value for the random generator.

Returns
No return value.

See Also
getRandom, getRandomSeed

VectorAdd(vecA , vecB)

Purpose
Use the ***VectorAdd*** function to add two vectors of up to three elements each to each other

Syntax
vecA – A vector of up to three elements.
vecB – A vector of up to three elements.

Returns
Returns the result of ***vecA*** + ***vecB***.

See Also
vectorSub

VectorCross(vecA , vecB)

Purpose
Use the ***VectorCross*** function to calculate the cross product of two vectors of up to three elements each.

Syntax
vecA – A vector of up to three elements.
vecB – A vector of up to three elements.

Returns
Returns the result of ***vecA*** x ***vecB***.

Notes
Remember, the resultant vector will always be an right angles to both input vectors.

See Also
VectorDot

VectorDist(vecA , vecB)

Purpose
Use the ***VectorDist*** function to calculate distance between two vectors of up to three elements each.

Syntax
vecA – A vector of up to three elements.
vecB – A vector of up to three elements.

Returns
Returns the result of " |*Xa* – *Xb*| |*Ya* – *Yb*| |*Za* – *Zb*| ".

See Also
VectorLen

VectorDot(vecA , vecB)

Purpose
Use the ***VectorCross*** function to calculate the dot product of two unit vectors of up to three elements each.

Warning! Be sure to always normalize both vecA and vecB before attempting to find the dot product. Calculating a dot product using un-normalized vectors (non- unit vectors) will result in meaningless results.

Syntax
vecA – A unit-vector of up to three elements.
vecB – A unit-vector of up to three elements.

Returns
Returns a scalar value equal to the result of ***vecA . vecB.*** This value which will always be a single floating-point value in the range [-1 , +1].

Notes
If the return value is < 0, the inner-angle between the vectors is > 90 degrees.
If the return value is == 0, the inner-angle between the vectors is == 90 degrees.
If the return value is > 0, the inner-angle between the vectors is < 90 degrees.

See Also
VectorCross

VectorLen(vec)

Purpose
Use the ***VectorLen*** function to calculate the length of vector ***vec***.

Syntax
vec - A vector of up to three elements.

Returns
Returns a scalar representing the length of the vector vec.

See Also
VectorDist

VectorNormalize(vec)

Purpose
Use the ***VectorNormalize*** function to calculated the unit vector equivalent of the vector ***vec***.

Syntax
vec - A vector of up to three elements.

Returns
Returns the unit vector equivalent of the vector ***vec***.

See Also
VectorScale

VectorOrthoBasis(vec)

Purpose
Use the ***VectorOrthoBasis*** function to calculate a 3x3 Row-Major formated matrix representing the orthogonal basis for the vector ***vec***.

Syntax
vec - A four element vector of the form "AxisX AxisY AxisZ theta", where theta is the angle of rotation about the vector formed by the prior three values.

Returns
Returns a 3x3 Row-Major formated matrix.

VectorScale(vec , scale)

Purpose
Use the ***VectorScale*** function to scale the vector ***vec*** by the scalar ***scale***.

Syntax
vec - A vector of up to three elements.
scale - A numeric value (integer or floating-point) representing the scaling factor.

Returns
Returns a scaled version of the vector **vec,** equivalent to:

```
" ( scale * X ) ( scale * Y ) ( scale * Z ) "
```

See Also
VectorNormalize

VectorSub(vecA , vecB)

Purpose
Use the ***VectorSub*** function to subtract ***vecB*** from ***vecA***.

Syntax
vecA - Left side vector in subtraction equation.
vecB - Right side vector in subtraction equation.

Returns
Returns a new vector equivalent to: "***vecA* - *vecB***"

See Also
vectorAdd

A.4 GUI Controls Quick Reference

A.4.1. Purpose

This appendix has been created to facilitate scripted creation and use of the standard Torque GUI controls. With the exclusion of the GuiControl profile class, all GUI classes with specific fields, methods, and/or callbacks are listed alphabetically, not by function.

A.4.2. GuiControlProfile Fields

Th GUIControlProfile (profile) class serves a similar purpose to that played by SimDataBlock, except in the context of GUI controls. Instances of this class are used to initialize common features of GUI controls. All GUI controls require a profile.

Creation Syntax

```
new GuiControlProfile ( GUIProfileName [ : parentProfile ] )
{
    field_0 = value;
...
    field_N = value;
...
    [dynamicfield_N = value;]
};
```

Usage Syntax

Normally, the profile for a GUI control is selected out of the profile pulldown in the GUI Inspector, but they can also be set and changed directly via script:

```
new GUIControl( GUIControlName ) {
    profile = GUIProfileName;
    // ...
};

// subsequently, it can be changed:
GUIControlName.profile = GUIProfileName2;
```

Fields

GUIControlProfiles provide the following fields. Not all fields are used by all controls, nor are all fields interpretted the same by all controls that use them. Several examples of field useage are given in the **GPGT** GUI Sampler.

Field Name	Type	Description
autoSizeHeight	Bool	Auto-size the height-bounds of the control to fit it's contents.
autoSizeWidth	Bool	Auto-size the width-bounds of the control to fit it's contents.
bitmap	String	Location of this control's bitmap.
border	ColorI	For most controls, if border is > 0 a border will be drawn, some controls use this to draw different types of borders however.
borderColor	ColorI	Border color, used to draw a border around the bounds if border is enabled.
borderColorHL	ColorI	Used instead of borderColor when the object is highlighted.
borderColorNA	ColorI	Used instead of borderColor when the object is not active or disabled.
borderThickness	Integer	Thickness of border in pixels.
canKeyFocus	Bool	True if the object can be given keyboard focus (in other words, made a first responder).
cursorColor	ColorI	Color for the blinking cursor in text fields (among other cases).
fillColor	ColorI	Fill color, this is used to fill the bounds of the control if it is opaque.
fillColorHL	ColorI	This is used instead of fillColor if the object is highlighted.
fillColorNA	ColorI	This is used instead of fillColor if the object is inactive or disabled.
fontColor	ColorI	Color of base font. "\c0".
fontColorHL	ColorI	Color of highlighted font. "\c1".
fontColorLink	ColorI	This is used as the font color for embedded URLs.
fontColorLinkHL	ColorI	This is used as the font color for embedded URLs when they are clicked or otherwise highlighted.
fontColorNA	ColorI	Color of inactive font. "\c2".
fontColors[10]	ColorI	Each of these corresponds to a font color "\cn". Note: Instead of using fontColors for the first four, they should be specified using fontColor, fontColorHL, fontColorNA, and fontColorSel, in that order.
fontColorSEL	ColorI	Color of selected font. "\c3".
fontSize	Integer	Size of font. Points or Pixels?
fontType	String	Font face name for the control.
hasBitmapArray	bool	Enables skinning in skinnable controls
justify	String	Justification for text: • left • center • right
modal	Bool	If set to true, this is a modeless dialog meaning it will pass input through instead of taking it all.
mouseOverSelected	Bool	If set to true, this control should be "selected" while the mouse is over it. (Only used by guiTextListCtrl)
numbersOnly	Bool	If true and a text control, this control should accept only numeric inputs.
opaque	Bool	If true, this control is not translucent.
returnTab	Bool	Used in GuiTextEditCtrl to specify if a tab-event should be simulated when return is pressed.
soundButtonDown	AudioProfile	Sound played when the object is "down" i.e. a button is pushed.
soundButtonOver	AudioProfile	Sound played when the mouse is over the object.
tab	Bool	This control is accessible via the tab key. (i.e. can be tabbed to).

Field Name	Type	Description
textOffset	Vector	Vector of two integers "x-offset y-offset".

A.4.3. Standard GUI Controls (Alphabetical Listing)

GuiBitmapBorderCtrl

This skinnable control is used to adorn other controls with a frame (or border). If you have never designed a GUI skin, see Standard GUIs chapter in Tech School Section of **GPGT**.

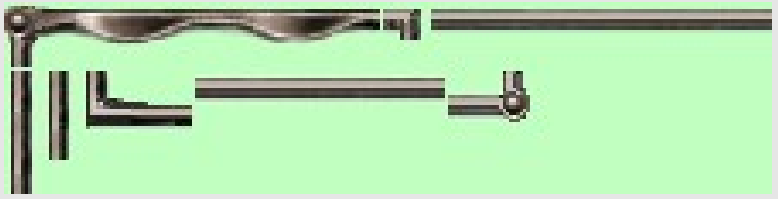
Skinning

- Define a profile with the following settings:

```
new GuiControlProfile ( aProfileName )
{
    // ...
    hasBitmapArray = true;
    bitmap         = "path to bitmap array graphic";
};
```

- Provide an image file with the following structure:

Column 0	Column 1	Column 2	Column 3	Column 4
Upper-left border	Upper-right border	Top border	--	
Left border	Right Border	Lower-left border	Lower border	Lower-right border



GuiBitmapButtonCtrl

This control is a skinnable button. Unlike other skinned controls, this control takes a maximum of four normal (non-array) graphics. Graphics files for this control use the following naming convention:

```
prefix_tag.suffix
```

- **prefix** – Any name for the image file.
- **_tag** – Any of the following (based on button state):
  - **_n** – Normal
  - **_h** – Highlighted
  - **_d** – Depressed
  - **_i** – Inactive
- **suffix** – png, jpg, bmp, etc.

For example, we could provide the following four images:

gglogo_n.png
(normal)

gglogo_h.png
(highlighted)

gglogo_d.png
(depressed)

gglogo_i.png
(inactive)

If an image file is not provided for any of the states: highlighted, depressed, or invalid, the normal image will be substituted. The normal image is always required. Also, setting the extent to "0 0" in the GUI Inspector and then pressing apply will cause the GUI to expand to the size of the image file.

Fields

Field Name	Description	Sample
bitmap	Path to image file, must be of the format: "path\path\path\prefix" **Do not provide tag or suffix**	".\gglogo"

Console Methods

setBitmap()

setBitmap(*pathName*)

Purpose
Use the setBitmap method to change the bitmap this control uses.

Syntax
pathName - A path to a new texture for this control.

Returns
No return value.

GuiBitmapCtrl

This control is used to display a small bitmap. In TGE versions prior to 1.4, this control can only accept a bitmap with a maximum size of 256 x 256 pixels. For larger images or in case of malfunction, use the GUIChunkedBitmapCtrl control.

Fields

Field Name	Description	Sample
bitmap	Path to image file. (Suffix is optional)	".\gg_background.png"
wrap	Boolean value enabling wrapping. If wrap is **false** and the image is larger than the GUIBitmapCtrl extent, the image will be down-scaled. Vice-versa, if the extent is larger than the image it will be scaled up. However, if wrap is **true**, and the extent is smaller than the image, then only a portion of the image will be visible.	[false , true]

Console Methods

setBitmap()	**setValue()**

setBitmap(*pathName*)

Purpose
Use the ***setBitmap*** method to change the bitmap this control uses.

Syntax
pathName - A path to a new texture for this control.

Returns
No return value.

`setValue( xOffset , yOffset );`

Purpose
Use the ***setValue*** method modify the positioning of the bitmap this control displays.

Syntax
xOffset – The pixel x-offset for the upper-left corner of this control's bitmap.
yOffset – The pixel y-offset for the upper-left corner of this control's bitmap.

Returns
No return value.

GuiButtonBaseCtrl

This is the base class to all other buttons and should **NOT** be used to make buttons. Its only job is to provide common fields and methods for the GuiBitmapButtonCtrl, GuiButtonCtrl, GuiCheckBoxCtrl, and GuiRadioCtrl.

Fields

Field Name	Description	Sample
buttonType	Each button has a default type, but that type can be over-ridden to be: - PushButton – Normal on/off button - ToggleButton – On/Off button that toggles. - RadioButton – When grouped with other radio buttons, only one button in the group may be on. groupNum should be specified when using this setting.	PushButton ToggleButton RadioButton
groupNum (radio buttons only)	Provide a positive integer greater than or equal to zero to group a set of radio buttons. i.e. to group three radio buttons, give them each the same groupNum.	-1 - Ungrouped [0 , inf) - Grouped
text	Text to be displayed on button. Not displayed on GuiBitmapButtonCtrl buttons with a valid image.	"Cancel"

Console Methods

getText	performClick	setText

`getText()`

Purpose
Use the ***getText*** method to get the current value of this control's text field.

Returns
Returns the current value of the text field for this control.

See Also
setText

performClick()

Purpose
Use the ***performClick*** method to force a click-event for this control.

setText(newText)

Purpose
Use the ***newText*** method to set the curent value of this control's text field.

Syntax
newText - A string containing text to replace the text field's old value with.

Returns
No return value.

GuiButtonCtrl

The standard button. It defaults to a buttonType of PushButton. All functionality comes from its parent GuiBaseButtonCtrl.

GuiCanvas

The canvas is the owner of all controls. That is, all other controls are placed in the The canvas can always be found for scripting purposes using the name: 'Canvas'. In order to display another control, that control must be made the 'content' of the Alternately, dialogs, can be pushed onto a canvas layer. Only dialogs exist in layers. The dialog in the highest layer always receives the keyboard and mouse input, unless the dialog is modeless. In this case, the dialog will only capture the input if it is the focus, else the inputs will all through to the next layer. Note: Modal dialogs supersede all other controls below them when it comes to capturing input.

Console Methods

cursorOff	cursorOn	getContent	getCursorPos
hideCursor	isCursorOn	popDialog	popLayer
pushDialog	renderFront	repaint	reset
setContent	setCursor	setCursorPos	showCursor

cursorOff()

Purpose
Use the ***cursorOff*** method to disable the cursor.

Returns
No return value.

cursorOn()

Purpose
Use the ***cursorOn*** method to enable the cursor.

Returns
No return value.

getContent()

Purpose
Use the ***getContent*** method to get the ID of the control which is being used as the current canvas content.

Returns
Returns the ID of the current canvas content (a control), or 0 meaning the canvas is empty.

getCursorPos()

Purpose
Use the ***getCursorPos*** method to retrieve the current position of the mouse pointer.

Returns
Returns a vector containing the "x y" coordinates of the cursor in the canvas.

hideCursor()

Purpose
Use the ***hideCursor*** method to hide the cursor.

Returns
No return value.

isCursorOn()

Purpose
Use the ***isCursorOn*** method to see if the cursor is currently enabled.

Returns
Returns true if cursor is on, false otherwise.

popDialog(***handle*** **)**

Purpose
Use the ***popDialog*** method to remove a currently showing dialog. If no handle is provided, the top most dialog is popped.

Syntax
handle – The ID or a previously pushed dialog.

Returns
No return value.

See Also
pushDialog, popLayer

popLayer(***layer*** **)**

Purpose
Use the ***popLayer*** method to remove (close) all dialogs in the specified canvas **'*layer*'**.

Syntax
layer – A integer value in the range [0 , inf) specifying the canvas layer to clear.

Returns
No return value.

See Also
pushDialog, popDialog

pushDialog(***handle*** **[,** ***layer*** **])**

Purpose
Use the ***pushDialog*** method to open a dialog on a specific canvas ***layer***, or in the same layer the last openned dialog. Newly placed dialogs placed in a layer with another dialog(s) will overlap the prior dialog(s).

Syntax
handle – The numeric ID or name of the dialog to be opened.
layer – A integer value in the range [0 , inf) specifying the canvas layer to place the dialog in.

renderFront(***enable*** **)**

Purpose
Use the ***renderFront*** method to modify the canvas rendering order.

Syntax
enable – A boolean value. If true, layers are rendered front-to-back, otherwise, the are rendered back-to-front (default).

See Also
popDialog, popLayer

repaint()

Purpose
Use the ***repaint*** method to force the canvas to redraw all elements.

Returns
No return value.

reset()

Purpose
Use the ***reset*** method to reset the current canvas update region.

Returns
No return value.

setContent(*handle*)

Purpose
Use the ***setContent*** method to set the control identified by ***handle*** as the current canvas content.

Syntax
handle – The numeric ID or name of the control to be made the canvas contents.

Returns
No return value.

setCursor(*cursorHandle*)

Purpose
Use the ***setCursor*** method to select the current cursor.

Syntax
cursorHandle – The ID of a previously defined GuiCursor object.

Returns
No return value.

setCursorPos( )

Purpose
Use the ***setCursorPos*** method to set the position of the cursor in the cavas.

Syntax
position – An "x y" position vector specifying the new location of the cursor.

showCursor()

Purpose
Use the ***showCursor*** method to enable the display of the cursor.

Returns
No return value.

Gotchas

- Do not try to run two copies of TGE in stand-alone mode unless the binary is complied to Debug. There is code in release compiles that prevents two or more copies of the canvas from running simultaneously on a **Windows platform**. You are probably safe on OSX and Linux, but if you encounter errors opening multiple instances of TGE, this may be the cause.
- Do not [ever] try to create a second instance of the Canvas within the same running image of TGE. **You will crash or hang your game**.

GUICheckBoxCtrl

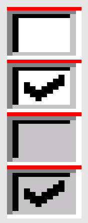
This skinnable control displays the perennial check-box control. By default this control toggles between on and off. If you have never designed a GUI skin, see Standard GUIs chapter in Tech School Section of **GPGT**.

Skinning

- Define a profile with the following settings:

```
new GuiControlProfile ( aProfileName )
{
    // ...
    hasBitmapArray = true;
    bitmap         = "path to bitmap array graphic";
};
```

- Provide an image file with the following structure:

Sample Array Image	Column 0
	Unchecked Normal
	Checked Normal
	Unchecked Inactive
	Checked Inactive

GuiChunkedBitmapCtrl

This control is the big brother to GuiBitmapCtrl and serves the same purpose. Its main value is that it can handle images larger than 256 x 256.

Fields

Field Name	Description	Sample
bitmap	Path to image file. File extension is optional.	".\ggbackground"
tile	Boolean value enabling wrapping. If tile is **false** and the image is larger than the GUIBitmapCtrl extent, the image will be down-scaled. Vice-versa, if the extent is larger than the image it will be scaled up. However, if tile is **true**, and the extent is smaller than the image, then only a portion of the image will be visible.	[false , true]
useVariable	See External Bitmap Specification below.	[false , true]

External Bitmap Specification

The GuiChunkedBitmapCtrl provides an interesting feature. It is possible to leave the bitmap field empty and to tell the control to get its bitmap from a global variable. To do this, specify your control similarly to this:

```
new GuiChunkedBitmapCtrl()
{
    // ...
    useVariable = true;
    variable    = "MyBitmap";
    bitmap      = "";
};
```

Of course, for this to work we must have defined $MyBitmap:

```
$MyBitmap = ".\some\path\to\some\image";
```

GuiControl

GUI Control is the root class to all GUI controls and thus provides many fields and console methods. When it is used as a control, it is normally used as a container.

Fields

Member (Field)	Description	Sample/Range
accelerator	• Specifies the hot-key for this command. See key mappings index.	--
altcommand	• Specifies a command(s) to be executed on a specific control action. • This field is used in a control-specific way.	echo("this is the altCommand");
command	• Specifies a command(s) to be executed on a specific control action. • This field is used in a control-specific way.	echo("this is the command");
extent	• These two values specify the pixel "WIDTH HEIGHT" of the control.	--
horizSizing	• In short, this field affects re-sizing and re-positioning of controls in relation to resolution.	right, width, left, center, relative
minExtent	• These two values specify the minimum size "WIDTH HEIGHT" of the control. • Some controls provide an alternate/supplemental field that does something similar.	--
modal	• Setting this does **NOTHING**. (Deprecated field)	false
position	• This value specifies the coordinate of the control's upper-left corner.	"0 0"
profile	• This allows you to select a predefined profile for your control. • Profiles are used to provide default field values for controls. • Values you supply in the inspector will over-ride profile defined values. • A value of <NULL> means, "use no profile". • Most profiles are defined the file: *example\common\ui\defaultProfiles.cs*. • To fine all defined profiles, search for '*new ControlProfile*' in all .CS files.	SomeGUIProfile

Member (Field)	Description	Sample/Range
setFirstResponder	• Setting this does **NOTHING**. (Deprecated field)	false
variable	• This field is used to specify the name of a (global) console variable which will be updated with the value of the control. • This field is used in a control-specific way.	"MyGlobalVar"
vertSizing	• In short, this field affects re-sizing and re-positioning of controls in relation to resolution.	bottom, height, top, center, relative
visible	• Specifies whether the GUI should start off in the visible state, or hidden.	[false , true]

Console Methods

getExtent	getMinExtent	getPosition	getValue
isActive	isAwake	isVisible	makeFirstResponder
resize	setActive	setProfile	setValue
setVisible			

getExtent()

Purpose
Use the ***getExtent*** method to determine the extent of the current control.

Returns
Returns a two-value integer vector containing the "x y" extent of the control.

getMinExtent()

Purpose
Use the ***getMinExtent*** method to determine the minimum allowed extent of this control.

Returns
Returns a two-value integer vector containing the "x y" minimum allowed extent of the control.

getPosition()

Purpose
Use the ***getPosition*** method to get the position of the upper-left corner of this control.

Returns
Returns a two-value integer vector containing the "x y" position of the control's upper-left corner.

getValue()

Purpose
Use the ***getValue*** method to get the control-specific 'value' for this control.

Returns
Returns a control-specific specific value. Varies by control.

isActive()

Purpose
Use the ***isActive*** method to determine if this control is active.

Returns
Returns true if this control is active.

Notes
An inactive control may visible, but will not accept inputs. It will also normally re-shade or re-skin itself to reflect its inactive state.

isAwake()

Purpose
Use the ***isAwake*** method to determine if this control is awake.

Returns
Returns true if this control is awake and ready to display.

isVisible()

Purpose
Use the ***isVisible*** method to determine if this control is visible.

Returns
Returns true if the control is visible.

Notes
This can return true, even if the entire control covered by another. This merely means that the control will render if not covered.

makeFirstResponder(***isFirst*** **)**

Purpose
Use the ***makeFirstResponder*** method to force this control to become the first responder.

Syntax
isFirst - A boolean value. If true, then this control become first reponder and
at captures inputs before all other controls, excluding dialogs above
this control.

Returns
No return value.

resize(***ulX*** **,** ***ulY*** **,** ***width*** **,** ***height*** **)**

Purpose
Use the ***resize*** method to resize and/or re-position a control. The upper-left corner of the control will be placed at <**uliX**, ***ulY***> in its parent and the control will be given an extent of "***width height***".

Syntax
ulX - The upper-left X coordinate of this control in pixels.
ulY - The upper-left Y coordinate of this control in pixels.
width - The width of this control in pixels.
height - The height of this control in pixels.

Returns
No return value.

setActive(***isActive*** **)**

Purpose
Use the ***setActive*** method to (de)activate this control. Once active, a control can accept inputs. Controls automatically re-shade/skin themselves to reflect their active/inactive state.

Syntax
isActive - A boolean value. f ***isActive*** is true, this control is activated, else it is set to inactive.

Returns
No return value.

setProfile(*profileName*)

Purpose
Use the ***setProfile*** method to change this control's profile to ***profileName***.

Syntax
profileName – A previously defined control profile.

Returns
No return value.

setValue(*value*)

Purpose
Use the ***setValue*** method to set the control specific value to ***value***. Purpose and type varies by control type.

Syntax
value – Some control specific value.

Returns
No return value.

setVisible(*isVisible*)

Purpose
Use the ***setVisible*** method to (un)hide this control.

Syntax
isVisible – A boolean value. If true, the control will be made visible, otherwise the control will be hidden.

Returns
No return value.

Callbacks

onAction()	onAdd()	onRemove	onSleep()
onWake()			

onAction(theControl)

Purpose
This generic callback will fire if there is an action event and no value is specified for the command field.

Syntax
theControl – The ID of this control.

onAdd(theControl)

Purpose
The ***onAdd*** callback is called when this control is created.

Syntax
theControl – The ID of this control.

onRemove(theControl)

Purpose
This callback fires when the control is destroyed.

Syntax
theControl – The ID of this control.

onSleep(theControl)

Purpose
This callback fires when the control is removed from the canvas, as a result of a call to setContent(***control***), where ***control*** is not this control. This is also called when the control is destroyed, prior to calling onRemove().

Syntax
theControl – The ID of this control.

onWake(theControl)

Purpose
This callback fires when the control is added to the canvas via the setContent(***control***) method call, where ***control*** is this control.

Syntax
theControl – The ID of this control.

GuiCursor

TGE allows us to define our own cursors, using a simple image file and some information defining the location of the cursor's hot-spot. In order to use a custom cursor, tell the canvas to activate it using the setCursor() method.

Fields

Field Name	Description	Sample
bitmapName	Path to cursor image.	".\mycursor.png"
hotSpot	A two-element vector specifying the offset from the image's upper-left corner where the hot-spot of the cursor should be located.	"4 4" Offset by 4 pixels in x and y.

GuiFadeInBitmapCtrl

This control is used to display an image by fading it in, then out over specified times. This control can be made to fade in and out continously, by putting it to sleep at the end of the fade out cycle and then waking it back up. This has to be done with a script and a call to schedule.

Fields

Field Name	Description	Sample
done	Boolean value denoting that this control is done fading out.	**Always initialize as false.**
fadeInTime	Integer value specifying time to fade in (in milliseconds).	[0 , inf)
fadeOutTime	Integer value specifying time to fade out (in milliseconds).	[0 , inf)
waitTime	Integer value specifying time to wait after fade-in completes, before starting to fade out.	[0 , inf)

Callbacks

click

```
click( theControl )

Purpose
This callback is fired, if the control has the focus and either a mouse button is
clicked, or a keyboard key is pressed.

Syntax
theControl - The ID of this control.
```

GUIFilterCtrl

This odd control allows us to specify a multi-knotted spline-like GUI that can be used to create a vector of floating-povalues (one per knot), where each value is between 0.0 and 1.0. The control can be used both as an input device and as a feedback device (we can set the position of each knot from script).

Fields

Field Name	Description	Sample
controlPoints	Number of knots to use.	3
filter	A floating-povector containing the default values for each knot. Values are bracketed between: [0.0 , 1.0]	"0.25 0.5 0.75"

Console Methods

getValue()	identity()	setValue()

getValue()

Purpose
Use the ***setVisible*** method to (un)hide this control.

Syntax
isVisible – A boolean value. If true, the control will be made visible, otherwise the control will be hidden.

Returns
No return value.

Returns a n-tuple floating-povector of values for each knot (left to right).

identity()

Purpose
Resets the filter and places all knots on a line following a 45 degree angle from 0.0 (bottom) on the left to 1.0 (top) on the right.

Returns
No return value.

setValue(knots)

Purpose
Sets the knot value for each knot by position.

Syntax
knots – a vector containing the knot positions for each knot. Each knot has a value between 0.0 and 1.0..

Returns
No return value.

GuiInputCtrl

This control is used to capture all input events. Input events in this case are such things as mouse clicks and/or keystrokes. For every input event, a callback is fired.

Callbacks

onInputEvent()

```
onInputEvent( theControl , deviceString , actionString , makeOrBreak )

Purpose
 Is called on make or break actions for all input devices.  A make action for the mouse
being a click-press, and a break action being the click-release.

Syntax
   theControl – The ID of this control.
 deviceString – "keyboard", "mouse", "mouse", etc. (see devices strings below).
 actionString – Event + modifiers (see actions strings below).
  makeOrBreak – Was this a make or break action [ 0 , 1 ]?
```

Device Strings		
keyboard	mouse	joystick
keyboard0 .. N	mouse0 .. N	joystick0 ..N

Note: If there are multiple devices of the same type, specify an instance number. i.e. keyboard1. If there is only one instance, just using an instance number is acceptable, but not required.

Keyboard Actions + Modifiers			
a .. z	A .. Z	F1 .. F24	0 .. 1
backspace	tab	return	enter
shift	ctrl	alt	pause
capslock	escape	space	pagedown
pageup	end	home	left
up	right	down	print
insert	delete	help	win_lwindow (win)
win_rwindow (win)	win_apps (win)	cmd (mac)	opt (mac)
lopt (mac)	ropt (mac)	numpad0 .. numpad9	numpadmult
numpadadd	numpadsep	numpadminus	numpaddecimal
numpaddivide	numpadenter	numlock	scrolllock
lshift	rshift	lcontrol	rcontrol
lalt	ralt	tilde	minus
equals	lbracket	rbracket	backslash
semicolon	apostrophe	comma	slash
lessthan	exclamation	grave	greaterthan

Joystick/Mouse Actions
button .. 31

Mouse Actions			
xaxis	yaxis	zaxis	rxaxis
ryaxis	rzaxis	slider	

Joystick POV Actions			
xpov	ypov	upov	dpov
lpov	rpov	xpov2	ypov2
upov2	dpov2	lpov2	rpov2

Miscellaneous Actions	
anykey	nomatch

```
function myControl::onInputEvent( %this, %device, %action, %makeOrBreak )
{
    echo( "** onInputEvent called - device = ", %device,
          ", action = ", %action,
          ", makeOrBreak = ", %makeOrBreak, " **" );
}
```

GUIMenuBar

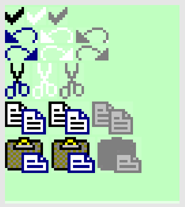
This semi-skinnable control displays the familiar menu bar control. By semi-skinnable, it is meant that graphic icons can be embedded in menu items, but the bar and the drop-downs themselves are not skinnable. If you have never designed a GUI image array, see Standard GUIs chapter **GPGT**.

Menu Item Icon Arrays

- Define a profile with the following settings:

```
new GuiControlProfile ( aProfileName )
{
    // ...
    hasBitmapArray = true;
    bitmap         = "path to bitmap array graphic";
};
```

- Provide an image file with the following structure:

Sample Array Image	Column 0	Column 1	Column 2
	Checked Mark	Not-Checked Mark	Inactive Checked Mark
	Optional Icon 0 (on)	Optional Icon 0 (off)	Optional Icon 0 (inactive)
	…	…	…
	Optional Icon N (on)	Optional Icon N (of)	Optional Icon N (inactive)

In effect, a GUIMenuBar can have any number of icon row, but the first (0th) row is normally reserved for the 'checked' icons. You can of course use any icon for that you wish, and you can use those icons elsewhere too.

GUIMenuBar Guidelines/Rules

The following guidelines/rules apply when building menus:

1. Build and Place – Place and size the initial menu bar using the GUI Editor.
2. Populate – Open the .gui file (or use a separate .cs) and write code to populate the menu.
3. Menus' and Menu Items' names should not start with a digit
4. Menu Items may optionally have accelerators
5. Menu Items may be enabled and disabled from script.
6. Menu Items may have separator lines (-----) between them.
7. Menus' and MenuItems' text can be dynamically changed from scripts.
8. Menu Items can be hidden.
9. Menu Items can have check box behavior and radio behaviors, including the display of a currently checked image in-menu.
10. Menus and Menu Items can be identified/referred to either by Menu/Menu Item text or ID.
11. Hierarchical (cascading) menus are not supported.
12. Menus do not support accelerators (only Menu Items support this)

Console Methods

Note: In the descriptions below, 'Menu Name' and 'Menu Item Name' refer to the Menu and Menu Item text respectively.

addMenu	**addMenuItem**	**clearMenuItems**	**clearMenus**
removeMenu	**removeMenuItem**	**setMenuItemBitmap**	**setMenuItemChecked**
setMenuItemEnable	**setMenuItemText**	**setMenuItemVisible**	**setMenuText**
setMenuVisible			

addMenu(menuName , menuID)

Purpose
Adds a new menu to the menu bar.

Syntax
menuName - The text (name) of the new menu entry.
  menuID - The ID of the new menu entry.

Returns
No return value.

addMenuItem(menuID | menuName , menuItemName , menuItemID ,
[accelerator] , [checkGroup])

Purpose
Adds a sub-menu entry to the specified menu.

Syntax
menuID – The ID of the menu.
menuName – The text (name) of the menu.
menuItemID – The ID of the menu item.
menuItemName – The text (name) of the menu item.
accelerator – A boolean value. If set to true, the sub-menu entry is checked, otherwise it is unchecked.
checkGroup – The check group this item should belong to, if any.

Returns
No return value.

clearMenuItems(menuID | menuName)

Purpose
Removes all the sub-menu items from the specified menu.

Syntax
menuID – The ID of the menu.
menuName – The text (name) of the menu.

Returns
No return value.

clearMenus()

Purpose
Clears all menus and sub-menus from the menu bar.

Returns
No return value.

removeMenu(menuID | menuName)

Purpose
Removes the specified menu from the menu bar.

Syntax
menuID – The ID of the menu.
menuName – The text (name) of the menu.
menuItemID – The ID of the menu item.
menuItemName – The text (name) of the menu item.
checked – A boolean value. If set to true, the sub-menu entry is checked, otherwise it is unchecked.

Returns
No return value.

removeMenuItem(menuID | menuName , menuItemID | menuItemName)

Purpose
Removes the specified menu item from the menu.

Syntax

menuID	– The ID of the menu.
menuName	– The text (name) of the menu.
menuItemID	– The ID of the menu item.
menuItemName	– The text (name) of the menu item.

Returns
No return value.

setMenuItemBitmap(menuID | menuName , menuItemID | menuItemName , bitmapIndex)

Purpose
Sets the specified menu item bitmap index in the bitmap array. Setting the item's index to -1 will remove any bitmap.

Syntax

menuID	– The ID of the menu.
menuName	– The text (name) of the menu.
menuItemID	– The ID of the menu item.
menuItemName	– The text (name) of the menu item.
bitMapIndex	– An integer value specifying the row of bitmap entries to use for sub-menu entry.

Returns
No return value.

setMenuItemChecked(menuID | menuName , menuItemID | menuItemName , checked)

Purpose
Sets the menu item bitmap to a check mark, which must be the first element in the bitmap array. Any other menu items in the menu with the same check group become unchecked if they are checked.

Syntax

menuID	– The ID of the menu.
menuName	– The text (name) of the menu.
menuItemID	– The ID of the menu item.
menuItemName	– The text (name) of the menu item.
checked	– A boolean value. If set to true, the sub-menu entry is checked, otherwise it is unchecked.

Returns
No return value.

setMenuItemEnable(menuID | menuName , menuItemID | menuItemName , enabled)

Purpose
Sets the menu item to enabled or disabled.

Syntax
menuID – The ID of the menu.
menuName – The text (name) of the menu.
menuItemID – The ID of the menu item.
menuItemName – The text (name) of the menu item.
enabled – A boolean value. If set to true, the sub-menu entry is enabled, otherwise it is disabled.

Returns
No return value.

setMenuItemText(menuID | menuName , menuItemID | menuItemName , newMenuItemText)

Purpose
Sets the text of the specified menu item to the new string.

Syntax
menuID – The ID of the menu.
menuName – The text (name) of the menu.
menuItemID – The ID of the menu item.
menuItemName – The text (name) of the menu item.
newMenuItemText – The new text for the specified sub-menu entry.

Returns
No return value.

setMenuItemVisible(menuID | menuName, menuItemID | menuItemName, visible)

Purpose
Use the **setMenuItemVisible** method to enable or disable the visibility of a specific sub-menu entry.

Syntax
menuID – The ID of the menu.
menuName – The text (name) of the menu.
menuItemID – The ID of the menu item.
menuItemName – The text (name) of the menu item.
visible – A boolean value. If set to true, this sub-menu entry will be shown, otherwise it will be hidden.

Returns
No return value.

setMenuText(menuID | menuName , newMenuText)

Purpose
Sets the text of the specified menu to the new string.

Syntax
```
     menuID - The ID of the menu.
   menuName - The text (name) of the menu.
newMenuText - The new text to give the menu entry.
```

Returns
No return value.

setMenuVisible(menuID | menuName , visible)

Purpose
Use the **setMenuVisible** method to enable or disable the visibility of a specific menu entry.

Syntax
```
  menuID - The ID of the menu.
menuName - The text (name) of the menu.
 visible - A boolean value.  If set to true, this menu entry will be shown,
           otherwise it will be hidden.
```

Returns
No return value.

Callbacks

When a menu is clicked it calls the onMenuSelect() method before displaying the drop-down menu items list. This allows the callback to enable/disable menu items, add/remove menu items, etc. in a context-sensitive way.

When a menu item is clicked, the drop-down menu items list removes itself from the display, then calls the onMenuItemSelect() method.

onMenuItemSelect	onMenuSelect

onMenuItemSelect(theControl , menuID , menuName , menuItemID , menuItemName)

Purpose
Called when a menu item is selected. Provides the menu's ***menuID***, and the text in the menu ***menuName***, as well as menu item's ***menuItemID***, and the text in the menu Item ***menuItemName***.

Syntax
```
  theControl - The ID of this control.
      menuID - The ID assigned to this menu item.
    menuName - The text assigned to this menu item.
  menuItemID - The ID assigned to this sub-menu item.
menuItemName - The text assigned to this sub-menu item.
```

onMenuSelect(theControl , menuID , menuName)

Purpose
Called when a menu is selected. Provides the menu's ***menuID***, and the text in the menu ***menuName***.

Syntax
```
theControl - The ID of this control.
    menuID - The ID assigned to this menu item.
  menuName - The text assigned to this menu item.
```

Returns
No return value.

GuiMessageVectorCtrl

This control is normally used to build a chat hud, but it can be used for a number of other purposes as well. In order to use this control, a MessageVector object must also be used (see MessageVector below). Since the actual data to be displayed is stored in the MessageVector and not this control, we can remove and add GUIMessageVectorCtrl controls at will and not corrupt the message data. This control is capable of displaying colorized text and can support beyond the base ten colors. Additionally, this control will recognize URLs and supports opening an external browser on url-click events.

Fields

Field Name	Description	Sample/Range
allowedMatches[16]	This string(s) is used to match a message string. When we have a positive match, the matched string will be highlighted with matchColor color. Up to 16 strings can be matched (watched for).	allowedMatches[0] = "**GPGT**"; allowedMatches[1] = "http";
lineContinuedIndex	TBD	integer
lineSpacing	Spacing between lines in pixels.	[0 , inf)
matchColor	Highlight color to use for strings matched against allowedMatches[n].	"128 255 255"
maxColorIndex	This index can be used to restrict or increase the number of supported text colors	[0 , 99]

Console Methods

attach	detach

attach(aVector)

Purpose
Make this gui control display messages from the specified MessageVector.

Syntax
aVector – A previously created messageVector instance.

Returns
No return value.

detach()

Purpose
Stop listening to messages from the MessageVector this control was previously attached to.

Returns
No return value.

Callbacks

urlClickCallback

urlClickCallback(theControl , *url*)

Purpose
Called when a URL is clicked in this control.

Syntax
theControl – The ID of this control.

MessageVector

The MessageVector object is a container of text meant to be consumed by the GuiMessageVectorCtrl control. Having said that, this class can be used for various other text storage purposes too.

Message Vector Console Methods

clear	**deleteLine**	**dump**	**getLineIndexByTag**
getLineTag	**getLineText**	**getLineTextByTag**	**getNumLines**
insertLine	**popBackLine**	**popFrontLine**	**pushBackLine**
pushFrontLine			

clear()

Purpose
Clear the message vector.

Returns
No return value.

deleteLine(lineIndex)

Purpose
Delete the line at the specified position.

Syntax
lineIndex – The line to delete in this vector.

Returns
No return value.

See Also
insertLine, pushBackLine, pushFrontLine

dump(filename [, header])

Purpose
Dump the message vector to a file, optionally prefixing the file with a ***header***.

Syntax
filename – The file to dump this vector to.
header – An optional string containing data to dump to the new file prior to dumping the vector.

Returns
No return value.

getLineIndexByTag(tag)

Purpose
Scan through the vector, returning the line number of the first line that matches the specified tag; else returns -1 if no match was found.

Syntax
tag – A special marker, possibly embedded in one or more lines in the vector.

Returns
Returns the line number of the first line found with the specified tag, otherwise returns -1.

See Also
insertLine, pushBackLine, pushFrontLine

getLineTag(line)

Purpose
Use the ***getLineTag*** method to retrieve the tag for the specified line.

Syntax
line – Line to search for tag in.

Returns
Returns a tag value or 0 indicating no tag found.

See Also
insertLine, pushBackLine, pushFrontLine

getLineText(index)

Purpose
Use the ***getLineIndex*** method to get the text at a specified line.

Syntax
index – The index in the vector from which to retrieve a line of text.

Returns
Returns the text at the specified line, or "" indicating a bad index.

See Also
insertLine, pushBackLine, pushFrontLine

getLineTextByTag(tag)

Purpose
Use the ***getLineTextByTag*** method to scan through the lines in the vector, returning the first line that has a matching tag.

Syntax
tag – An special marker that may have been used when creating lines in the vector.

Returns
Returns the contents of the first line found with a matching tag, or "" indicating no match.

See Also
insertLine, pushBackLine, pushFrontLine

getNumLines()

Purpose
Use the ***getNumLines*** method to get the number of lines in the vector.

Returns
Returns an integer value equal to the line count for this vector.

See Also
insertLine, pushBackLine, pushFrontLine

insertLine(pos , msg [, tag])

Purpose
Use the ***insertLine*** method to insert a new line into the vector at the specified position. An optional tag may also be applied.

Syntax
pos - The line at which to insert the new text.
msg - The text to add to this control.
tag - An optional tag to tag this line with. If not tag is supplied, a tag of 0 is used.

Returns
No return value.

See Also
pushBackLine, pushFrontLine

popBackLine()

Purpose
Use the ***popBackLine*** method to pop a line from the back of the list; destroys the line.

Returns
No return value.

See Also
insertLine, pushBackLine, pushFrontLine

popFrontLine()

Purpose
Use the ***popFrontLine*** method to pop a line from the front of the vector, destroying the line.

Returns
No return value.

See Also
insertLine, pushBackLine, pushFrontLine

pushBackLine(msg [, tag])

Purpose
Use the ***pushBackLine*** method to push a line onto the back of the list.

Syntax
msg - The text to add to this control.
tag - An optional tag to tag this line with. If not tag is supplied, a tag of 0 is used.

Returns
No return value.

See Also
popBackLine, popFrontLine, insertLine, pushFrontLine

pushFrontLine(msg [, tag])

Purpose
Use the ***pushFrontLine*** method to push a line onto the front of the vector.

Syntax
msg - The text to add to this control.
tag - An optional tag to tag this line with. If not tag is supplied, a tag of 0 is used.

Returns
No return value.

See Also
popBackLine, popFrontLine, insertLine, pushBackLine

See Also
popBackLine, popFrontLine, insertLine, pushBackLine

GuiMLTextCtrl

This control is a multi-line markup-language supporting text control (ML == Markup Language). In addition to printing multi-line text, this control will accept TGE Markup-Language (TorqueML) formatted text, allowing us to make changes to the font, font-weight, color, etc. A complete listing of the TorqueML tokens and the Syntax for using them is supplied below. This control also supports onURL() and onResize() callbacks.

Fields

Field Name	Description	Sample/Range
allowColorChars	Enable colored text.	[false , true]
deniedSound	Audio profile played when current number of characters == maxChars and an attempt is made to add new characters.	Audio Profile
lineSpacing	Integer value specifying number of pixels between lines.	2
maxChars	Integer value specifying number of characters that will fit in this control.	[0 , inf)
text	Initial text to dispaly in control.	"Type Here"

Console Methods

addText()	**forceReflow()**	**getText()**	**scrollToTag()**
setAlpha()	**setCursorPosition()**	**setText()**	

addText(***text*** **,** ***reformat*** **)**

Purpose
Use the ***addText*** method to add new text to the control. You may optionally request that the control be reformatted.

Syntax
```
    text – Text to add to control.
reformat – A boolean value that when set to true forces the control to
           re-evaluate the entire contents and to redisplay it.
```

Returns
No return value.

See Also
getText, setText, forceReflow

forceReflow()

Purpose
Use the ***forceReflow*** method to force the text control to re-evaluate the entire contents and to redisplay it, possibly resizing the control.

Returns
No return value.

See Also
addText

getText()

Purpose
Use the ***getText*** method to return the current text contents of the control, including all formatting characters.

Returns
Returns the entire text contents of the control or "" indicating no contents.

See Also
addText

scrollToTag(tagID)

Purpose
Use the ***scrollToTag*** method to scroll to the first instance of a tag if it exists.

Syntax
tagID – A tag number to search for. These tags are specified by embedding TorqueML <tag:tag_number> entries in text.

Returns
No return value.

See Also
scrollToTop, setCursorPosition

scrollToTop()

Purpose
Use the ***scrollToTop*** method to scroll to the top of the text.

Returns
No return value.

See Also
scrollToTag, setCursorPosition

setAlpha(*alpha*)

Purpose
Use the ***setAlpha*** method to set ***alpha*** of this control to between [0.0 , 1.0].

Syntax
alpha - A floating point value between 0.0 and 1.0 indicating the control's new alpha setting.

setCursorPosition(newPos)

Purpose
Use the ***setCursorPosition*** method to offset the cursor by ***newPos*** characters into the current text contents of the control.

Syntax
newPos - An integer value indicating the character position at which to place the cursor.

Returns
No return value.

See Also
scrollToTag, scrollToTop

setText(text)

Purpose
Use the ***setText*** method to change the current text content of the control to ***text***. This replaces all old content.

Syntax
text - The new contents for this control.

Returns
No return value.

See Also
addText, getText

Callbacks

onURL()	**onResize()**

onURL(theControl , url)

Purpose
This callback is called when a hyperlink or a gamelink is clicked in the control.

Syntax
theControl - The ID of this control.
url - The url value that was specified by the TorqeML link statement.

```
Notes:
If a normal URL was specified like this:
          <a:gamers.hallofworlds.com>name</a>
, the value of url will be the string "gamers.hallofworlds.com"

If a gamelinke URL was specified (starts with the string "gamelink") like this:
             the <a:gamelink_SomeTopic>name</a>
, the value of url will be the string "gamelink_SomeTopic"
```

onResize(theControl , width , height)

```
Purpose
This calleback is fired when the control is resized.

Syntax
theControl - The ID of this control.
     width - The new width for this control.
    height - The new height for this control.
```

TGE Markup Language (TorqueML)

Markup Tag	Purpose	Samples
Fonts and Text Effects		
<font:***font_name*** :***font_size***>	Changes font type and size. ***font_name*** – Any legal font name. ***font_size*** – [1 , 32] points	<font:Arial Bold:16> <font:Verdana Italic:32> <font:Palatino Linotype:4> <font:Lucida Console:10>
<color:***hex_tag***>	Changes subsequent text color to that represented in ***hex_tag***.	<color:FF0000> Pure Red <color:00FFCC > A Nice Green
<shadow:***XOffset***:***YOffset***>	Enables shadowed text with an offset of "XOffset YOffset" pixels.	<shadow:10:10> down 10, right 10 <shadow:-5:-10> up 5, left 10
<shadowcolor:***hex_tag***>	Changes subsequent text shadow color to that represented in ***hex_tag***.	<color:C0C0C0> Light Gray
Test Justification		
<just:left>	Justifies text to left.	<just:left>
<just:right>	Justifies text to right.	<just:right>
<just:center>	Justifies text to center.	<just:center>
Text Clipping		
<clip:***pixels***>***text***</clip>	Forces ***text*** to be clipped if is over ***pixels*** pixels wide. Clipped text has ... appended to the end.	<clip:50>abcdefghijk</clip> Displays -> **abcde...** <clip:20>abcdefghijk</clip> Displays -> **a...**
Margins		
<lmargin%:***percent***>	Sets left margin to ***percent*** percentage of viewable area.	<lmargin%:5> <lmargin%:20>
<rmargin%:***percent*** >	Sets right margin to ***percent*** percentage of viewable area.	<rmargin%:5> <rmargin%:20>
<lmargin:***pixels***>	Sets left margin to ***pixels*** number of	<lmargin:12>

Markup Tag	Purpose	Samples
	pixels.	<lmargin:100>
<rmargin:***pixels***>	Sets right margin to ***pixels*** number of pixels.	<rmargin:12> <rmargin:100>
HyperLinks		
<a:***link_address***>***text***</a>	Creates a hyper-text link to ***link_address*** and displays ***text***.	<linkcolor:0000FF><a:www.hallofworlds.com>HOW</a> Displays -> HOW
<a:*gamelink****TAG***>***text***</a>	Same as-<a:***link_address***> but address is not underlined. Can be used for making context sensitive help GUIs and other non-web links.	<linkcolor:0000FF><a:gamelink**Topic0**>Topic 0</a> Displays -> Topic 0
<linkcolor:***hex_tag***>	Changes subsequent hyperlink color to that represented in ***hex_tag***.	<linkcolor:0000FF> For Blue Links
Images		
<bitmap:***path/filename***>	Displays a bitmap. Note: path/filename must be a complete path, starting at the mod directory.	<bitmap:egt/client/ui/gglogo.png>
Tables		
<tab:***tabstop***,**tabstiop**,...,***tabstop***>	Use to create table like entities with columns **tabstop** pixels wide. Subsequently, Each new line is a table row. Columns are separated by tabs.	<tab:100,150,100> Subsequent lines should be treated as a three column table with columns 100, 150, and 100 pixels wide respectively.
Attribute Stacking		
<spush>	Saves current attributes on stack.	<spush>
<spop>	Pop last stacked attributes, restoring them.	<spop>
<sbreak>	**Do not use.**	--
Miscellaneous		
<br>	Line break.	Hello<br>World
<div:>		
<tag:***tag_number***>	Places a numeric tag ***tag_number*** in this line. This is used for searching and is not displayed.	<tag:100>

GuiMLTextEditCtrl

This control is a TorqueML formatted text entry. Nearly all of its functionality derives from its parent GuiMLTextCtrl (above).

Fields

Field Name	Description	Sample/Range
escapeCommand	Command to execute on escape key press (while editting).	--

GuiMouseEventCtrl

This control is used to capture and react to (via callback) all standard mouse events. Most controls do not automatically execute a callback for all mouse events. This helps reduce the possible flood of callbacks that would otherwise occur in complicated GUIs. However, when an event needs to be captured, and the control in question does not already do so, simply add this control as a child of the control needing to capture the event and be sure it covers the areas where the event should be captured. This control will then capture the following events:

- Left/Right Mouse Down
- Left/Right Moue Up
- Mouse Move
- Left/Right Mouse Drag
- Mouse Enter
- Mouse Exit

, with these (possible) modifiers:

- Left/Right/Either Shift
- Left/Right/Either Control
- Left/Right/Either ALT

Fields

Field Name	Description	Sample
lockMouse	If true, this control temporarily ignores the mouse.	[false , true]

Globals

The following globals are made available for script writting purposes.

Field Name	Description
$EventModifier::LSHIFT	Left Shift-Key Depressed
$EventModifier::RSHIFT	Right Shift-Key Depressed
$EventModifier::SHIFT	Either Shift-Key Depressed
$EventModifier::LCTRL	Left Ctrl-Key Depressed
$EventModifier::RCTRL	Right Ctrl-Key Depressed
$EventModifier::CTRL	Either Ctrl-Key Depressed
$EventModifier::LALT	Left Alt-Key Depressed
$EventModifier::RALT	Right Alt-Key Depressed
$EventModifier::ALT	Either Alt-Key Depressed

Callbacks

onMouseDown()	onMouseDragged()	onMouseEnter()	onMouseLeave()
onMouseMove()	onMouseUp()	onMouseRightDown()	onMouseRightDragged()
onMouseRightUp()			

onMouse*(theControl , eventModifier , XY , numMouseClicks)

Purpose
Called when the * event occurs.

All onMouse* events take the same modifiers. The possibilities for ***** are:

- ***Down*** – Left mouse button-pressed.
- ***Dragged*** – Mouse moved while left-button held down.
- ***Enter*** – Mouse entered area covered by this control.
- ***Leave*** - Mouse left area covered by this control.
- ***Move*** - Mouse moved in area covered by this control
- ***Up*** – Left mouse button-released.
- ***RightDown*** – Right mouse button-pressed.
- ***RightDragged*** – Mouse moved while right-button held down.
- ***RightUp*** – Right mouse button-released.

As noted above, this callback will be passed the ***eventModifier***(s) listed in globals(above). These modifiers are bit-masks and should be treated as such.

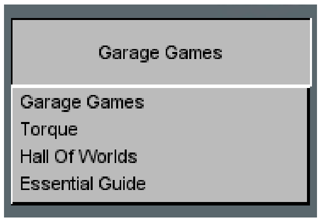
Lastly, the double-click time is 500ms, thus if the mouse is clicked two or more times in that period of time, this callback will get a numMouseClicks > 1. This is used primarily to determine if a double-click or a single-click processing should occur.

Syntax

theControl	– The ID of this control.
eventModifier	– One or more of the globals specified above.
XY	– The position of the mouse in the current control.
numMouseClicks	– The number of clicks recorded in the last 500ms.

GuiPopUpMenuCtrl

This is a traditional pop-up menu. When a left-mouse click is applied to this control, a list will pop up. This list will either be above or below the control dependent on its placement, how many entries are in the list, and the nearness of the bottom of its parent. In the case that the list is taller than the height of its parent or maxPopupHeight, it will scroll automatically. Additionally, each text entry can be themed with a coloring scheme.



Fields

Field Name	Description	Sample/Range
maxPopupHeight	Integer value specifying the maximum number of list entries to show at any one time. List scrolls if there are more entries than maxPopupHeight.	--

Console Methods

add()	addScheme()	clear()	findText()
forceClose()	forceOnAction()	getSelected()	getText()
getTextByID()	replaceText()	setEnumContent()	setSelected()
setText()	size()	sort()	

add(*entryText* , *entryID* [, *entryScheme*])

Purpose
Use the ***add*** method to add a new entry with text ***entryText***, ID ***entryID***, and using the scheme ***entryScheme***.

Syntax

entryText	– Text to display in menu entry.
entryID	– ID to assign to entry. This value may be 1 or greater.
entryScheme	– An integer value representing the ID of an optional color scheme to be used for this entry.

Returns
No return value.

See Also
addScheme, clear

addScheme(***entryScheme*** **,** ***fontColor*** **,** ***fontColorHL*** **,** ***fontColorSEL*** **)**

Purpose
Use the ***addScheme*** method to create a new color scheme or to modify an existing one.

Syntax

entryScheme	– An integer value representing the ID of the scheme, between 1 and inf.
fontColor	– A vector containing an integer representation of the menu entry's normal color.
fontColorHL	– A vector containing an integer representation of the menu entry's highlighted color.
fontColorSEL	– A vector containing an integer representation of the menu entry's selected color.

Returns
No return value.

Notes
An integer color vector contains three integer values, each between 0 and 255 and is organized in this order: "R G B",

See Also
add

clear()

Purpose
Use the ***clear*** method to remove all entries and schemes from the menu.

Returns
No return value.

See Also
add, addScheme

findText(***text*** **)**

Purpose
Use the ***findText*** method to locate the string ***text*** in the list of menu items. It will return the ID of the first entry found.

Syntax
text – A string containing the text to search for.

Returns
Returns an integer value representing the ID of the menu item, or -1 if the text was not found.

Notes
This is an exact match, so if the menu item is "Gish" and you search for "Gis", or "gish', or any other variation that does not match the entire menu item and the case of each letter, the search will not find a match.

forceClose()

Purpose
Use the ***forceClose*** method to force the menu to close.

Returns
No return value.

Notes
This is useful for making menus that fold up after a short delay when the mouse leaves the menu area.

See Also
forceOnAction

forceOnAction()

Purpose
Use the ***forceOnAction*** method to force the *onAction* callback to be triggered.

Returns
No return value.

See Also
forceClose, onAction (GUIControl callback)

getSelected()

Purpose
Use the ***getSelected*** method to get the ID of the last selected entry.

Returns
Returns the ID of the currently selected entry, or 0 meaning no menu was selected after the last menu open.

Warning
If someone opens and then closes the menu without making a selection, the selected entry goes back to 0, even if an entry was previously selected.

See Also
getText, setSelected

getText()

Purpose
Use the ***getText*** method to get the text currently displayed in the menu bar.

Returns
Returns the text in the menu bar or “” if no text is present.

See Also
getSelected, setText

getTextById(***ID*** **)**

Purpose
Use the ***getTextById*** method to get the text value for the menu item represented by ***ID***.

Syntax
ID – An integer value representing the ID of a menu item.

Returns
Returns a string containing the menu item corresponding to ID, or a NULL string ("") if no menu item has the specified ID.

See Also
add, getText

replaceText(***enable*** **)**

Purpose
Use the ***replaceText*** method to enable the updating of the menu bar text when a menu item is selected.

Syntax
enable – A boolean value enabling or disabling the automatic updating of the menu bar text when a selection is made.

Returns
No return value.

Notes
This does not prevent changing the menu bar text with setText.

See Also
getText, setText

setEnumContent(***className*** **,** ***enumName*** **)**

Purpose
Use the ***setEnumContent*** method to fill the menu with a class reps field enumerations.

Syntax
className – The class name associated with this enum content.
enumName – The name of the enumerated entry to add to the menu. This value must match an enum string as exposed by the engine for the class. The menu item will have the same text as the enum string name, and the ID will be equal to the enumerated entries value.

Returns
No return value.

setSelected(***ID*** **)**

Purpose
Use the ***setSelected*** method to force the selection of a specific entry in the menu as identified by ***ID***.

Syntax
ID – An integer value representing the ID of the entry to select.

Returns
No return value.

Notes
This will cause the menu text to update and the *onSelect* callback will fire.

If the control is currently in no-replace mode, the text in the menu bar will not be updated by this selection.

See Also
getSelected, onSelect (callback), replaceText, setText

setText(***text*** **)**

Purpose
Use the ***setText*** method to change the text displayed in the menu bar.

Syntax
text – New text to display in the menu bar.

Returns
No return value.

Notes
Pass the NULL string (“”) to clear the menu bar text.

See Also
getText, replaceText

size()

Purpose
Use the ***size*** method to determine the number of entries in the menu.

Returns
Returns an integer value representing the number of menu items currently in the menu.

See Also
add

sort()

Purpose
Use the ***sort*** method to sort the menu in ascending order.

Returns
No return value.

Notes
This is a lexicographic sort, so number (1,2,3,...) will come before letters (a,b,c,...).

Callbacks

onCancel	onSelect

onSelect(theControl , *ID* , *text*)

Purpose
Called when an entry is selected from the list.

Syntax
theControl - The ID of this control.
ID - ID of entry selected.
text - Text for entry selected.

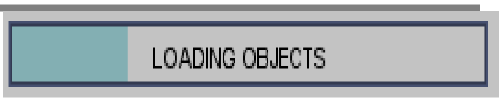
onCancel(theControl)

Purpose
Called when a selection fails. i.e. if a scripted selection (setSelected()) provides an invalid entry ID, this event will be called.

Syntax
theControl - The ID of this control.

Returns
No return value.

GuiProgressCtrl



This control is used to reflect a percentage. Furthermore, it is usually used to give feedback on the progress of a task. It does not add any new fields, methods, or callbacks and is derived from the GUIControl. To update the bar, simply use the setValue() methods with an argument between 0.0 and 1.0. Ex:

```
%myProgressBar.setValue(0.25); // Sets bar to 25% (left to right)
```

This control, does not have a text label, but this can easily be accomplished using a GuiTextCtrl.

GuiRadioCtrl

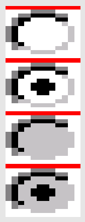
This is a skinnable radio button control. It is used when a group of buttons must have only one button set at any one time. If you have never designed a GUI skin, see Standard GUIs chapter in Tech School Section of **GPGT**.

Skinning

- Define a profile with the following settings:

```
new GuiControlProfile ( aProfileName ) {
    // ...
    hasBitmapArray = true;
    bitmap         = "path to bitmap array graphic";
};
```

- Provide an image file with the following structure:

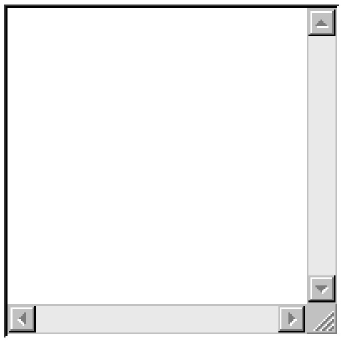
Sample Array Image	Column 0
	Unchecked Normal
	Checked Normal
	Unchecked Inactive
	Checked Inactive

In order for the radio control to behave properly, the buttons all need to have the same parent and groupNum. In this example, either "Radio 0" or "Radio 1" can be selected, but not both.

```
new guiControl() {
    new GuiRadioCtrl() {
        profile = "GuiRadioProfile";
        //..
        text = "Radio 0";
        groupNum = "1";
        buttonType = "RadioButton";
    };
    new GuiRadioCtrl() {
        profile = "GuiRadioProfile";
        //..
        text = "Radio 1";
        groupNum = "1";
        buttonType = "RadioButton";
    };
};
```

GuiScrollCtrl

This is a container control used in concert with a GUIMLText, GUIMLEditTextCtrl, GuiTextListCtrl, and other resizeable controls. These resizeable controls are made children of the GuiScrollCtrl, which then allows the user to use scroll bars to move to a specific location in the child contol. GuiScrollCtrl can be programmed to supply a vertical and/or a horizontal scrollbar. These scrollbars will be enabled (based on field settings), always, never, or when the child content expands beyond the vertical or horizontal bounds of the view area.

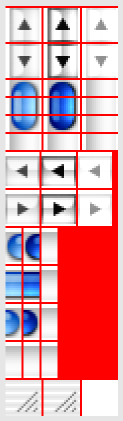


Skinning

- Define a profile with the following settings:

```
new GuiControlProfile ( aProfileName ) {
    // ...
    hasBitmapArray = true;
    bitmap         = "path to bitmap array graphic";
};
```

- Provide an image file with the following structure:

Sample Array Image	Column 0	Column 1	Column 2
	Up-Scroll Normal	Up-Scroll Depressed	Up-Scroll Inactive
	Down-Scroll Normal	Down-Scroll Depressed	Down-Scroll Inactive
	Top of Vertical Thumb Normal	Top of Vertical Thumb Depressed	Top of Vertical Thumb Inactive
	Middle of Vertical Thumb Normal	Middle of Vertical Thumb Depressed	Middle of Vertical Thumb Inactive
	Bottom of Vertical Thumb Normal	Bottom of Vertical Thumb Depressed	Bottom of Vertical Thumb Inactive
	Vertical Bar Normal	Vertical Bar Depressed	Vertical Bar Inactive
	Right-Scroll Normal	Right-Scroll Depressed	Right-Scroll Inactive
	Left-Scroll Normal	Left-Scroll Depressed	Left-Scroll Inactive
	Left of Horizontal Thumb Normal	Left of Horizontal Thumb Depressed	Left of Horizontal Thumb Inactive
	Middle of Horizontal Thumb Normal	Middle of Horizontal Thumb Depressed	Middle of Horizontal Thumb Inactive
	Right of Horizontal Thumb Normal	Right of Horizontal Thumb Depressed	Right of Horizontal Thumb Inactive
	Horizontal Bar Normal	Horizontal Bar Depressed	Horizontal Bar Inactive
	Lower-Right Affordance Normal	Lower-Right Affordance Depressed	Lower-Right Affordance Inactive

Fields

Field Name	Description	Sample
childMargin	A two-value integer vector specifying a horizontal and/or vertical offset for child entries. In effect this is a padding value.	"4 4" (Provide 4 pixels of padding.)
constantThumbHeight	If this is true, the bar drag affordance does not scale to reflect the number of entries in the list. This helps when the list is very full. It keeps the drag bar from getting too thin to be grabbed/selected.	[false , true]
hScrollBar	Attributes for horizontal bar.	alwaysOn alwaysOff dynamic
vScrollBar	Attributes for vertical bar.	alwaysOn alwaysOff dynamic
willFirstRepond	Boolean value enabling first responder status.	[false , true]

Console Methods

scrollToBottom()	scrollToTop()

scrollToTop()

Purpose
Use the ***scrollToTop*** method to scroll the scroll control to the top of the child content area.

Returns
No return value.

See Also
scrollToBottom

scrollToBottom()

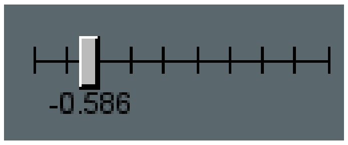
Purpose
Use the ***scrollToBottom*** method to scroll the scroll control to the bottom of the child content area.

Returns
No return value.

See Also
scrollToTop

GuiSliderCtrl

This is a numeric slider control. It allows a value between a lower and upper range to be selected using a sliding interface.



Fields

Field Name	Description	Sample
range	A two-element floating-point vector containing the the low and high values this control can take.	"-10.0 100.0" "min max"
ticks	Number of ticks to have on bar.	10
value	Initial value.	[min , max]

Console Methods

getValue()

getValue()

Purpose
Use the ***getValue*** method to get the current value of the slider.

Returns
Returns current value of control (position of slider).

Notes

1. Holding **SHIFT** while dragging slider selects closest tick.
2. If **altCommand** is specified, it will be called every sim tick while this control is awake.

GuiTextCtrl

This is a label control. It displays a fixed (256 characters or fewer) amount of text. It can be updated dynamically from script if needed.

Fields

Field Name	Description	Sample
maxLength	Integer limit on length of text in label. Max is 256. Text is truncated once it reaches the current limit.	25
text	Initial text for label.	"Hello world"

Console Methods

setText()

setText(***newText*** **)**

Purpose
Use the ***setText*** method to set the content of label to ***newText***.

Syntax
newText - A string representing the new value for this label.

Notes

1. If **altCommand** is specified, it will be called when the keypad-enter or regular enter-key is pressed.

GuiTextEditCtrl

This is a simple single-line text entry control. It is a child of GuiTextCtrl and is thus limited to a maximum of 256 characters and can be limited with the same mechanisms provided by its parent. This control can also recall prior entries and allows them to be recalled via the up and down arrows on the keyboard.

Fields

Field Name	Description	Sample
deniedSound	Audio profile for sound that should be played when typing continues and the entry is full or at its maxLength limit.	NoMoreSpaceSound
escapeCommand	Script to execute when escape key is pressed an this control is active.	--
historySize	Integer value determining number of prior entries to recall.	[0 , inf)
password	Boolean value specifying this is a password. If true, insted of letters, asterisks will be printed.	[false , true]
sinkAllKeyEvents	Boolean value specifying that this control will capture all key events. Note: This will be ignored of more than one GUITextEditCtrl within the same control has this set to true.	[false , true]
tabComplete	Boolean value specifying that when the tab key is pressed, the onTabComplete callback should be executed.	[false , true]
validate	Script that should be executed when this control loses focus.	--

Console Methods

getCursorPos()	**setCursorPos()**

getCursorPos()

Purpose
Use the ***getCursorPos*** method to get the current position of the text cursor in the control.

Returns
Returns the current position of the text cursor in the control, where 0 is at the beginning of the line, 1 is after the first letter, and so on.

See Also
setCursorPos

setCursorPos(newPos)

Purpose
Use the ***setCursorPos*** method to set the current position of text cursor to ***newPos***.

Syntax
newPos – The new position to place the cursor at, where 0 is at the beginning of the line, 1 is after the first letter, and so on.

Returns
No return value.

Notes
If the requested position is beyond the end of text, the cursor will be placed after the last letter. If the value is less than zero, the cursor will be placed at the front of the entry.

See Also
getCursorPos

Callbacks

onTablComplete

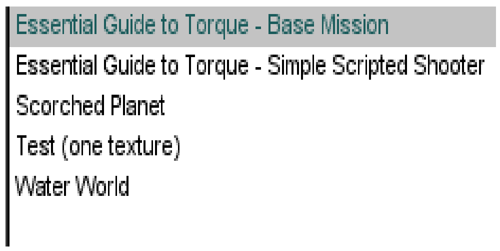
onTabComplete(theControl)

Purpose
This callback is executed if the tab key is pressed, and the tabComplete field is set to true.

Syntax
theControl – The ID of this control.

GuiTextListCtrl

This control is a mult-line scrollable list. Alone it can be used to display data, but in concert other controls (buttons), it can be used as a selection control. Futthermore, this can be made the child of a GuiScrollCtrl to allow for long lists.


Essential Guide to Torque - Base Mission
Essential Guide to Torque - Simple Scripted Shooter
Scorched Planet
Test (one texture)
Water World

Fields

Field Name	Description	Sample
clipColumnText	Boolean value instructing the control to clip contents of entries that extend beyond the visual edge of the current column.	[false , true]
columns	A vector contain a series of integer values corresponding to the pixel position for each column.	"0 50 100"

Field Name	Description	Sample
enumerate	No longer used.	--
fitParentWidth	Boolean value specifying that this control should expand to fit the extent (or other limits) of its parent.	[false , true]
resizeCell	No longer used.	--

Console Methods

addRow()	clear()	clearSelection()	findTextIndex()
getRowID()	getRowNumByID()	getRowText()	getRowTextByID()
getSelectedID()	isRowActive()	removeRow()	removeRowByID()
rowCount()	scrollVisible()	setRowActive()	setRowByID()
setSelectedByID()	setSelectedRow()	sort()	sortNumerical()

addRow(*ID* , *text* [, row])

Purpose
Use the ***addRow*** method to add a new entry to the text list.

Syntax

```
  ID - The integer ID to assign to this entry. May be between 0 and inf
       and can be repeated. i.e., multiple entries may have the same ID.
text - The text to display for this entry in the list.
 row - An optional integer value representing the position at which to
       add this entry, where 0 is the top of the list, 1 is after them
       first entry (if any), etc.
```

Returns
Returns the row number of the new entry.

See Also
clear, removeRow

clear()

Purpose
Use the ***clear*** method to remove all entries from the list.

Returns
No return value.

See Also
addRow, clearSelection, removeRow

clearSelection()

Purpose
Use the ***clearSelection*** method to deselect the current selection (if any).

Returns
No return value.

See Also
clear, setSelection

findTextIndex(text)

Purpose
Use the ***findTextIndex*** method to do an exact-match search for ***text*** in the list of items.

Syntax
text – The text to search for. Must match exactly or no match will occur.

Returns
No return value.

Notes
This is an exact match, so if the menu item is "Gish" and you search for "Gis", or "gish', or any other variation that does not match the entire menu item and the case of each letter, the search will not find a match.

See Also
getRowText, getRowTextByID

getRowId(row)

Purpose
Use the ***getRowId*** method to get the ID value for a specified ***row***.

Syntax
row – The row in the list to check the ID for.

Returns
Returns the ID of the specified row, or -1 if ***row*** is out of range.

Notes
Row numbers start at 0.

See Also
addRow, getRowNumByID, getRowText, getRowTextByID

getRowNumById(ID)

Purpose
Use the ***getRowNumById*** method to get the row number of the first entry in the list with the specified **ID**.

Syntax
ID – An integer value equal to an the entry ID to search for.

Returns
Returns the number of the first row found with a matching ***ID***, or -1 if no row contains the specified ***ID***.

Notes
Row numbers start at 0.

See Also
addRow, getRowID, getRowText, getRowTextByID

getRowText(row)

Purpose
Use the ***getRowText*** method to retrieve the text value of an entry in the list at the specified ***row***.

Syntax
row – The number of the list row from which to retrieve the text.

Returns
Returns the text found at the specified row, or the NULL string ("") if the ***row*** number is out of bounds.

Notes
Row numbers start at 0.

See Also
addRow, getRowID, getRowNumByID, getRowTextByID

getRowTextById(ID)

Purpose
Use the ***getRowTextById*** method to get the text of the first row with an ID matching the passed ***ID***.

Syntax
ID – An integer value equal to the entry ID to search for.

Returns
Returns a string containing the text of the first row with a matching ***ID***, or the NULL string ("") if no row matches the specified ***ID***.

See Also
addRow, findTextIndex, getRowID, getRowNumByID, getRowTextByID

getSelectedId()

Purpose
Use the ***getSelectedId*** method to return the ID value of the currently selected entry (if any).

Returns
Returns the integer ID of the currently selected row or -1 if no row is selected.

See Also
addRow, clearSelected, getRowID, getRowNumByID, getRowTextByID

isRowActive(row)

Purpose
Use the ***isRowActive*** method to determine if the specified ***row*** is active.

Syntax
row – The row to check the active status for.

Returns
Returns 1 if the row is active, or 0 if the row is inactive or the specified ***row*** is out of bounds.

Notes
Row numbers start at 0.

See Also
setRowActive

removeRow(row)

Purpose
Use the ***removeRow*** method to remove the specified ***row*** from the list.

Syntax
row – The number of the list row to be removed.

Returns
No return value.

Notes
Row numbers start at 0. Nothing is removed if ***row*** is out of bounds.

See Also
add, removeRowbyID

removeRowById(ID)

Purpose
Use the ***removeRowById*** method to remove the first row containing a matching ***ID***.

Syntax
ID – An integer value equal to the entry ID of the row to delete.

Returns
No return value.

See Also
add, removeRow, rowCount

rowCount()

Purpose
Use the ***rowCount*** method to determine how many entries are in the list.

Returns
Returns 0 if the list is entry or a positive integer value equal to the number of rows in the list if it is not empty.

See Also
add, removeRow

scrollVisible(row)

Purpose
Use the ***scrollVisible*** method to force the scrollList containing this text list to scroll so that the specified ***row*** is visible.

Syntax
row – The number of the list row to be scrolled to.

Returns
No return value.

Notes
Row numbers start at 0.

See Also
rowCount

setRowActive(row , active)

Purpose
Use the ***setRowActive*** method to activate or deactivate the specified ***row***.

Syntax
row - The number of the list row to activate or deactivate.
active - A boolean value specifying whether this row is active or inactive.

Returns
No return value.

Notes
Row numbers start at 0. The row will not change visibly, but we can check if a selected row is active later to determine whether to respond or not to this selection.

See Also
isRowActive

setRowById(*ID* , text)

Purpose
Use the ***setRowById*** method to update the ***text*** if the first row fond with an ID matching the specified ***ID***.

Syntax
ID - An integer value equal to the entry ID of the row to change the text for.
text - The text to replace the found row value with.

Returns
No return value.

setSelectedById(ID)

Purpose
Use the ***setSelectedById*** method to selected a row by a specified ***ID***. This will select the first row found to have an ID matching the specified ***ID***.

Syntax
ID - An integer value equal to the entry ID of the row to select.

Returns
No return value.

Notes
No selection will be made if no row has a matching ***ID***. Additionally, if no selection is made and a prior row was selected, that selection will stay in effect.

See Also
setSelectedRow

setSelectedRow(row)

Purpose
Use the ***setSelectedRow*** method to select a specified ***row*** in the list.

Syntax
row – The number of the list row to set as selected.

Returns
No return value.

Notes
Row numbers start at 0. No selection will be made if the ***row*** number is out of bounds. Additionally, if no selection is made and a prior row was selected, that selection will stay in effect.

See Also
setSelectedByID

sort(columnID [, ascending = false])

Purpose
Use the ***sort*** method to sort the list using a lexicographic sort. The sort order may be either ascending or descending (default).

Syntax
columnID – The column to sort on.
ascending – An optional boolean value, which when true means to do an ascending sort, otherwise the sort will be descending.

Returns
No return value.

Notes
Columns may be specified when setting up the list, by default most lists have one column so the ***columnID*** should be 0.

See Also
sortNumerical

sortNumerical(columnID [, ascending = false])

Purpose
Use the ***sortNumerical*** method to sort the list using a numeric sort. The sort order may be either ascending or descending (default).

Syntax
columnID – The column to sort on.
ascending – An optional boolean value, which when true means to do an ascending sort, otherwise the sort will be descending.

Returns
No return value.

Notes
Columns may be specified when setting up the list, by default most lists have one column so the ***columnID*** should be 0.

See Also
sort

Callbacks

onDeleteKey	onSelect

onDeleteKey(theControl , entry)

Purpose
This callback is executed if the delete key is pressed, and this control is active.

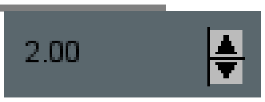
Syntax
theControl – The ID of this control.
entry – The ID of the currently selected text list item.

onSelect(theControl , entry)

Purpose
This callback is executed when an item in the text list is selected.

Syntax
theControl – The ID of this control.
entry – The text value of the selected text list item.

GuiTextEditSliderCtrl



This is another floating-poslider control, but it uses up-down buttons instead of a a left-right slider. This control is a bit more flexible in terms of its output as it uses a standard-C printf style formatting string.

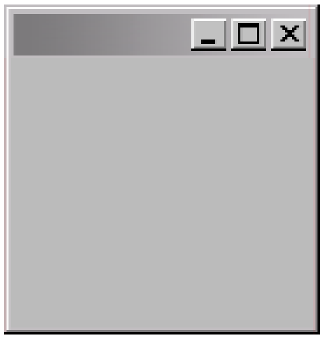
Fields

Field Name	Description	Sample
format	A format string of the form: %[Flags][Width].[Precision][Size][Type] See standard-C printf formatting rules for specifics. This is fed directly into an sprintf() command in TGE.	"%#5.5f"
range	A two-element floating-point vector containing the min/max range for this control.	"0.0 255.0" "min max"
increment	A floating-povalue specifying step size (per-click)	1

Notes

1. If **altCommand** is specified, it will be called with an argument of 'false' when the keypad enter or regular enter key is pressed.

GuiWindowCtrl



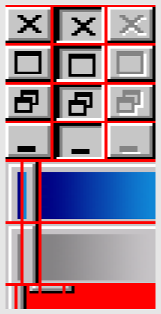
This is a completely skinnable window control. It behaves like a standard window, providing the ability to drag, resize, minimize, maximize, restore, and close. If you have never designed a GUI skin, see Standard GUIs chapter in Tech School Section of **GPGT**.

Skinning

- Define a profile with the following settings:

```
new GuiControlProfile ( aProfileName ) {
    // ...
    hasBitmapArray = true;
    bitmap         = "path to bitmap array graphic";
};
```

- Provide an image file with the following structure:

Sample Array Image	Column 0	Column 1	Column 2	Column 3	Column 4
	Close Button Normal	Close Button Depressed	Close Button Inactive	--	--
	Maximize Button Normal	Maximize Button Depressed	Maximize Button Inactive	--	--
	Revert Button Normal	Revert Button Depressed	Revert Button Inactive	--	--
	Minimize Button Normal	Minimize Button Depressed	Minimize Button Inactive	--	--
	Title Bar Left Edge	Title Bar Right Edge	Title Bar Middle	--	--
	Title Bar Left Edge Inactive	Title Bar Right Edge Inactive	Title Bar Middle Inactive	--	--
	Left Edge	Right Edge	Lower Left Corner	Bottom Edge	Lower Right Corner

Fields

Field Name	Description	Sample
canClose	Allow window to be closed.	[false , true]
canMaximize	Allow window to be maximized.If false, button does not render.	[false , true]
canMinimize	Allow window to be minimized. If false, button does not render.	[false , true]
canMove	Allow window to be moved.	[false , true]
closeCommand	Command(s) to issue on close.	[false , true]
minSize	Two-element integer vector specifying minimum <x,y> dimensions window can assume when drag-resizing.	"10 20"
resizeHeight	Enable (drag) height resizing.	[false , true]
resizeWidth	Enable (drag) width resizing.	[false , true]

GuiFrameSetCtrl

This control is used to automatically or manually frame any number of children controls, in regular row column format.

The first time you try to use it; it may seem a little odd, but once you understand the rules by which it operates, you'll be using it for all kinds of tasks.

Frames are organized as a grid, where frame 0 is the upper-left corner frame, frame N is the lower-right frame. Frame numbers increment left-to-right and top-to-bottom. i.e. A control with 3 rows and 3 frames would have these frame numbers:

```
0 |  1  | 2
------------
3 |  4  | 5
------------
6 |  7  | 8
```

Fields

Field Name	Description	Range/Sample
autoBalance	If set to true, the control will attempt to make all cells have the same height and width on waking.	[false , true]
borderColor	An integer color vector representing the color of the control's borders.	"255 128 128"
borderEnable	An enumerated string value determining if borders are rendered. This affects the filling of the border, not its presence. If you want no border, set borderWidth to 0.	alwaysOn alwaysOff dynamic
borderMovable	An enumerated string value determining if borders are mouse draggable.	alwaysOn alwaysOff dynamic
borderWidth	An integer value representing the pixel width of the borders.	[0 , inf]
columns	An integer vector specifying the pixel starting position of each new column.	"0 100 200"
fudgeFactor	An integer value representing the number of pixels to subtract from the ends of borders (making them shorter). Does not affect ability to grab borders.	[0 , inf]
rows	An integer vector specifying the pixel starting position of each new row.	"0 100 200"

Console Methods

addColumn()	**addRow()**	**frameBorder()**	**frameMinExtent()**
frameMovable()	**getColumnCount()**	**getColumnOffset()**	**getRowCount()**
getRowOffset()	**removeColumn()**	**removeRow()**	**setColumnOffset()**
setRowOffset()			

addColumn()

Purpose
Use the ***addColumn*** method to add another column to the control.

Returns
No return value.

Notes
The current contents of the GUIFrameCtrl may shift to fill the new column.

New columns are added on the right of the control.

See Also
addRow, removeColumn, removeRow

addRow()

Purpose
Use the ***addRow*** method to add another row to the control.

Returns
No return value.

Notes
The current contents of the GUIFrameCtrl may shift to fill the new row.

New rows are added on the bottom of the control.

See Also
addColumn, removeColumn, removeRow

frameBorder(index [, enable = true])

Purpose
Use the ***frameBorder*** method to change the frame's enable state.

Syntax
index – Frame index to enable/disable/
enable – Currently a boolean, but should actually be a string: alwaysOn, alwaysOff, dynamic.

Returns
No return value.

Notes
This function is not working as of this writing.

frameMinExtent(index, w, h)

Purpose
Use the ***frameMinExtent*** method to set the minimum extent allowed for a frame.

Syntax
index - The frame number
    w - Minimum width in pixels.
    h - Minimum height in pixels.

Returns
No return value.

Notes
These minimum extents do not prevent a parent control from collapsing the frame control and its frames. These limits apply to dragging and resizing as is done with the frames' draggable borders.

frameMovable(index [, enable = true])

Purpose
Use the ***frameMovable*** method to change the frame's draggable state.

Syntax
index - Frame index to enable/disable/
enable - Currently a boolean, but should actually be a string: alwaysOn, alwaysOff, dynamic.

Returns
No return value.

Notes
This function is not working as of this writing.

getColumnCount()

Purpose
Use the ***getColumnCount*** method to determine the number of columns in this control.

Returns
Returns an integer value equal to the number of columns in this frame.

See Also
getRowCount

getColumnOffset(column)

Purpose
Use the ***getColumnOffset*** method to determine the current pixel location of the specified ***column***.

Syntax
column – An integer value specifying the column to examine.

Returns
Returns the pixel offset for the specified ***column***.

Notes
Column 0 is the first column on the left side of frame 0. Column 1 is on the right side of frame 0 and the left side of frame 1, etc.

See Also
getRowOffset, setColumnOffset, setRowOffset

getRowCount()

Purpose
Use the ***getRowCount*** method to determine the number of rows in this control.

Returns
Returns an integer value equal to the number of rows in this frame.

See Also
getColumnCount

getRowOffset(row)

Purpose
Use the ***getRowOffset*** method to determine the current pixel location of the specified ***row***.

Syntax
row – An integer value specifying the row to examine.

Returns
Returns the pixel offset for the specified ***row***.

Notes
Row 0 is the first row on the top of the first row of frames. Row 1 is below the first row of frames and above the second row of frames, etc. 1, etc.

See Also
getColumnOffset, setColumnOffset, setRowOffset

removeColumn()

Purpose
Use the ***removeColumn*** method to remove a column from the right side of the control.

Returns
No return value.

Notes
Columns are removed right to left.

See Also
addColumn, addRow, removeRow

removeRow()

Purpose
Use the ***removeRow*** method to remove the bottom row from the control.

Returns
No return value.

Notes
Rows are removed bottom to top.

See Also
addColumn, addRow, removeColumn

setColumnOffset(column , offset)

Purpose
Use the ***setColumnOffset*** method to determine the current pixel location of the specified ***column***.

Syntax
column – An integer value specifying the column to examine.
offset – An integer value specifying the new column offset in pixels.

Notes
Column 0 is the first column on the left side of frame 0. Column 1 is on the right side of frame 0 and the left side of frame 1, etc.

The left-most and right-most columns cannot be moved.

See Also
getColumnOffset, getRowOffset, setRowOffset

setRowOffset(row , offset)

Purpose
Use the ***setRowOffset*** method to set the current pixel location of the specified ***row***.

Syntax
row - An integer value specifying the row to modify.
offset - An integer value specifying the new row offset in pixels.

Notes
Row 0 is the first row on the top of the first row of frames. Row 1 is below the first row of frames and above the second row of frames, etc. 1, etc.

The bottom-most and top-most rows cannot be moved.

See Also
getColumnOffset, getRowOffset, setColumnOffset

A.5 Callbacks Quick Reference

A.5.1. Game Callbacks

These of callbacks are the ones that you will encounter after the game has started.

aiPlayer::

onMoveStuck(*Obj*)

Purpose
Called when aiPlayer gets stuck while moving.

Syntax
Obj – The object this callback is called for.

onReachDestination(*Obj*)

Purpose
Called when aiPlayer reaches last programmed 'destination'.

Syntax
Obj – The object this callback is called for.

onTargetEnterLOS(*Obj*)

Purpose
Called when current target of Obj aiPlayer comes into it's field-of-view, and therefore line of sight.

Syntax
DB – The ID of the datablock this callback is executed on.
Obj – The object this callback is called for.

onTargetExitLOS(*Obj*)

Purpose
Called when aiPlayer loses sight of current target. comes

Syntax
DB – The ID of the datablock this callback is executed on.
Obj – The object this callback is called for.

gameBaseData::

onAdd(DB , Obj)

Purpose
Called when Obj object is added to the scene. This callback should be used to do any initialization work required for Obj object.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.

onNewDataBlock(DB , Obj)

Purpose
Called whenever a datablock needs to be registered on the server.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.

onRemove(DB , Obj)

Purpose
Called just before Obj object is removed from the scene. This callback should be used to do any cleanup work required for Obj object.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.

itemData::

onEnterLiquid(DB , Obj , percentCovered , liquidType)

Purpose
Called when an item enters water.

Syntax
```
            DB - The ID of the datablock this callback is executed on.
           Obj - The object this callback is called for.
percentCovered - A value between 0.0 and 1.0 equivalent to the water coverage.
    liquidType - Water         = 0,
                 OceanWater    = 1,
                 RiverWater    = 2,
                 StagnantWater = 3,
                 Lava          = 4,
                 HotLava       = 5,
                 CrustyLava    = 6,
                 Quicksand     = 7
```

onLeaveLiquid(DB , Obj , liquidType)

Purpose
Called when an item exits water.

Syntax
```
        DB - The ID of the datablock this callback is executed on.
       Obj - The object this callback is called for.
liquidType - Water         = 0,
             OceanWater    = 1,
             RiverWater    = 2,
             StagnantWater = 3,
             Lava          = 4,
             HotLava       = 5,
             CrustyLava    = 6,
             Quicksand     = 7
```

onStickyCollision(DB , Obj)

Purpose
Called when an item has a sticky collision.

Syntax
```
 DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.
```

lightningData::

applyDamage(DB , Obj , hitObject, hitPosition , hitNormal)

Purpose
Called when lightning strikes an object.

Syntax
```
         DB - The ID of the datablock this callback is executed on.
        Obj - The object this callback is called for.
  hitObject - ID of object struck by lightning.
hitPosition - Position of strike.
  hitNormal - Surface normal at strike position.
```

pathCamera::

onNode(Obj , node)

Purpose
Called when a path camera gets to a node on its path.

Syntax
```
 Obj - The object this callback is called for.
node - The number of the node that the camera got to.
```

playerData::

animationDone(DB , Obj)

Purpose
Called when a scripted animation (*playThread*) completes.

Syntax
 DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.

Notes
This doesn't specify which animation completed, just that an animation started by *playThread* completed.

doDismount(DB , Obj)

Purpose
Called when the player is mounted to another shape, and $mvTrigger2 is set to 1.

Syntax
 DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.

onEnterLiquid(DB , Obj , percentCovered, liquidType)

Purpose
Called when this player enters a waterblock.

Syntax

```
            DB - The ID of the datablock this callback is executed on.
           Obj - The object this callback is called for.
percentCovered - A value between 0.0 and 1.0 equivalent to the water coverage.
    liquidType - Water         = 0,
                 OceanWater    = 1,
                 RiverWater    = 2,
                 StagnantWater = 3,
                 Lava          = 4,
                 HotLava       = 5,
                 CrustyLava    = 6,
                 Quicksand     = 7
```

onEnterMissionArea(DB , Obj)

Purpose
Called when this player enters the mission area.

Syntax
 DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.

onLeaveLiquid(DB , Obj , liquidType)

Purpose
Called when this player exits a waterblock.

Syntax
```
 DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.
```

onLeaveMissionArea(DB , Obj)

Purpose
Called when this player leaves the mission area.

Syntax
```
 DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.
```

projectileData::

onCollision(colliderDB , colliderObject , collidedObj , fade , pos , normal)

Purpose
Called when a projectile strikes another object.

Syntax
```
    colliderDB - Datablock handle for Obj projectile object.
   colliderObj - Handle to Obj instance of the projectile object.
collidedObject - Handle to the instance of the object the projectile has struck.
          fade - How much Obj projectile has faded at the time of collision
                 [0.0 , 1.0].
           pos - World position the collision occurred.
        normal - The normal vector for the surface that was struck.
```

onExplode(DB , Obj , position , fade)

Purpose
Called when an projectile explodes.

Syntax
```
      DB - The ID of the datablock this callback is executed on.
     Obj - The object this callback is called for.
position - World position the collision occurred.
    fade - How much the projectile has faded at the time of explosion [0.0 , 1.0].
```

shapeBaseData::

onCollision(colliderDB , colliderObject , collidedObj , vec , speed)

Purpose
Called when a shape collides with an object.

Syntax
colliderDB - Datablock handle for Obj projectile object.
colliderObj - Handle to Obj instance of the projectile object.
collidedObject - Handle to the instance of the object the projectile has struck.
vec - The collision vector for Obj object.
speed - The velocity of the collision. Just the magnitude of the collision vector.

onDamage(DB , Obj , damage)

Purpose
The onDamage callback executes when the shape using this datablock receives damage or repair.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.
damage - Damage or repair that was applied to this shape.

onDestroyed(DB , Obj)

Purpose
Called when a shape is destroyed.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.

onDisabled(DB , Obj)

Purpose
Called when a shape is disabled.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.

onEnabled(DB , Obj)

Purpose
Called when a shape is enabled.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.

onEndSequence(DB , Obj , slot)

Purpose
Called when a non-cyclic animation ends for this shape.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.

onImpact(*colliderDB* , *colliderObject* , *collidedObj* , *vec* , *speed*)

Purpose
Called when a shape impacts an object.

Syntax
colliderDB - Datablock handle for Obj projectile object.
***collider*Obj** - Handle to Obj instance of the projectile object.
***collided*Object** - Handle to the instance of the object the projectile has struck.
vec - The collision vector for Obj object.
speed - The velocity of the collision. Just the magnitude of the collision vector.

onMount(DB , Obj , mountToObject , node)

Purpose
Called when this object mounts to another object.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.
mountToObject - The object this shape mounted to.
node - The node that this shape mounted to.

onTrigger(DB , Obj , triggerNum , triggerVal)

Purpose
Called when this object is told that a trigger was pressed or released.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.
triggerNum - Ths trigger number (between 0 and 6).
triggerVal - 1, if the trigger was pressed. 0, if the trigger was released.

onUnmount(DB , Obj , mountToObject , node)

Purpose
Called when this object dismounts from another object.

Syntax

```
           DB - The ID of the datablock this callback is executed on.
          Obj - The object this callback is called for.
mountToObject - The object this shape dismounted from.
         node - The node that this shape dismounted from.
```

shapeBaseImage::

onMount(DB , mountToObject , node)

Purpose
Called when this image mounts to a shape.

Syntax

```
           DB - The ID of the datablock this callback is executed on.
mountToObject - The object this image is mounted to.
         slot - The slot (for the shape this image is mounted to) this image
                is mounted in.
```

onUnmount(DB , mountToObject , node)

Purpose
Called when this image dismounts from a shape.

Syntax

```
           DB - The ID of the datablock this callback is executed on.
mountToObject - The object this image is mounted to.
         slot - The slot (for the shape this image is mounted to) this image
                is mounted in.
```

***userCallback*(DB , mountToObject , slot)**

Purpose
Called by an image's state machine. '*userCallback*' can be anything the user specifies in the state machine's 'userCallback' fields.

Syntax

```
           DB - The ID of the datablock this callback is executed on.
mountToObject - The object this image is mounted to.
         slot - The slot (for the shape this image is mounted to) this image
                is mounted in.
```

triggerData::

onEnterTrigger(DB , Obj , intruder)

Purpose
Called when an object enters the bounds of this trigger.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.
intruder - The object interacting with this trigger.

onLeaveTrigger(DB , Obj , intruder)

Purpose
Called when an object leaves the bounds of this trigger.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.
intruder - The object interacting with this trigger.

onTickTrigger(DB , Obj , intruder)

Purpose
Called when a trigger tick goes by and there are objects within the bounds of this trigger.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.
intruder - The object interacting with this trigger.

onTrigger(DB , Obj , enter)

Purpose
Executes when a group trigger enter or leave event occurs.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.
enter - If an object entered the group trigger, this value is 1, otherwise it is 0.

onTriggerTick(DB , Obj)

Purpose
Called when a group trigger ticks and objects are within the bounds of that trigger.

Syntax
DB - The ID of the datablock this callback is executed on.
Obj - The object this callback is called for.

vehicleData::

onEnterLiquid(DB , Obj , percentCovered , liquidType)

Purpose
Called when this vehicle enters a water block.

Syntax
```
            DB - The ID of the datablock this callback is executed on.
           Obj - The object this callback is called for.
percentCovered - A value between 0.0 and 1.0 equivalent to the water coverage.
    liquidType - Water        = 0,
                 OceanWater   = 1,
                 RiverWater   = 2,
                 StagnantWater = 3,
                 Lava         = 4,
                 HotLava      = 5,
                 CrustyLava   = 6,
                 Quicksand    = 7
```

onLeaveLiquid(DB , Obj , liquidType)

Purpose
Called when this vehicle leaves a water block.

Syntax
```
        DB - The ID of the datablock this callback is executed on.
       Obj - The object this callback is called for.
liquidType - Water        = 0,
             OceanWater   = 1,
             RiverWater   = 2,
             StagnantWater = 3,
             Lava         = 4,
             HotLava      = 5,
             CrustyLava   = 6,
             Quicksand    = 7
```

A.5.2. GUI Callbacks

Please see GUI Controls Quick Reference.

A.5.3. Other Callbacks

scriptObject::/scriptGroup::

onAdd(Obj)

Purpose
Called when a scriptObject or scriptGroup is created.

Syntax
Obj – The object this callback is called for.

onRemove(Obj)

Purpose
Called when a scriptObject or scriptGroup is about to be destroyed.

Syntax
Obj – The object this callback is called for.

Game

onExit()

Purpose
The onExit callback executes when the game starts to shutdown.

A.6 Scripted Systems Quick Reference

The EGTGE Kit comes with the following 'systems':

Volumes 1 & 2

- **Simple Task Manager** – This is a semi-advanced self-executing task management system that allows you to create a list of tasks. Tasks can be functions, statements, methods targeting some object, or some combination thereof. The task manager can then be told to start self-executing, or it can be manually forwarded, based on one's need. There is also a manager-manager that can be enabled to watch for rogue task managers, etc.
- **Simple Inventory** – This is a standard inventory system implemented using a scriptObject instead of in the current control object. This system is described thoroughly in Volume 1 Inventories.
- **EGTGE Utilities** – Although not actually a system, there are several utility functions and methods provided with the kit. Their use is documented here.

Next Book ONLY

- **EGTGE Brains** – This is a simple, yet very flexible, state-machine manager that can be be used to drive anything from complex user interfaces to AI players and vehicles.
- **EGTGE Ranged Weapons** – In the Weapons chapter of Volume 2, we discuss the difficult task of designing a set of weapons. This discussion leads to the definition and implementation of a weapons system designed to handle any type of mounted, thrown, or deployed ranged weapon.

This e-document contains references for each of the above systems. Please note, although this document is included in volume 1, the actual volume 2 systems are not available unless you have purchased that volume.

A.6.1. Simple Task Management System (SimpleTaskMgr)

While experimenting with the original TGE FPS kit, I came across some code embedded in the AI handling scripts. Basically, this code created a queue of tasks and then executed them. I thought, "Gee that would be useful for a variety of things. It sure would be nice if that functionality were centralized instead..." Thus, SimpleTaskMgr was born.

This 'task management system' can be used in both client and server space and provides the following features:

- Tasks can be enqueued in a 'task queue'
- A task queue can operate standalone (i.e. each task is a function), or a target object can be assigned to the task queue. In the latter case, each task in the queue (excluding special tasks) will be called on the target object: "%obj.sometask()".
- The tasks manager is an object and can be deleted, automatically canceling any pending tasks.
- Tasks can be programmed to recycle themselves (task executes and re-adds it self to end of queue between 0 and infinite times).
- Tasks can be programmed to preempt subsequent tasks (task executes and re-adds it self to front of queue a finite number of times).
- Tasks in the queue can be manually scheduled, or self-scheduled.
- When self-scheduling, tasks can provide their own schedule times, or use an overriding default time specified in the SimpleTaskMgr instance.
- There is no limit on the number of task manager that can be executing at once and all task managers are independent of all other managers.

This may seem a bit complicated, but be assured, the interfaces to the above features are simple and you don't need to use functionality you don't want. Furthermore, a thorough coverage of SimpleTaskMgr is provided below because it is used heavily in the EGTGE Kit.

SimpleTaskMgr Usage

Executing Functions

In its most basic incarnation, SimpleTaskMgr can be treated like a queue of tasks or function. In order to use it, we create a manger, add some tasks, execute the tasks, and remove the manager:

```
%testTaskMgr = newTaskManager();

%testTaskMgr.addTask( "echo(\"Hello\" );");
%testTaskMgr.addTask( "echo(\"World\" );");

%testTaskMgr.executeNextTask();
%testTaskMgr.executeNextTask();

%testTaskMgr.delete();
```

This would produce the output:

```
Hello
World
```

Executing Methods

In addition to executing a series of functions, we can tell the task manager to run a series of methods, by assigning a target object:

```
// A method to call on our target object
function TestObj::dummyTaskFunc( %this, %val) {
    echo("TestObj::dummyTaskFunc("@%val@")");
}

%myObj = new scriptGroup(TestObj);

%testTaskMgr = newTaskManager(%myObj);

%testTaskMgr.addTask( "dummyTaskFunc( 10 );" );
%testTaskMgr.addTask( "dummyTaskFunc( 20 );" );

%testTaskMgr.executeNextTask();
%testTaskMgr.executeNextTask();

%myObj.delete();
%testTaskMgr.delete();
```

This would produce the output:

```
TestObj::dummyTaskFunc(10);
TestObj::dummyTaskFunc(20);
```

Re-Targetting

Generally speaking, a SimpleTaskMgr instance should be used for either functions, or for methods, but not for both. However, an SimpleTaskMgr instance can be re-targeted at any time with the setTarget() method:

```
%testTaskMgr.setTarget( %someNewTarget );
```

Recycling

Because it would not be very useful to have to specify a repetitive list, or to re-specify a list that is supposed to run forever, SimpleTaskMgr supports the idea of recycling. A task can be added to the list with an additional argument specifying the number of times this task should 'recycled' before being removed from the list:

```
%testTaskMgr = newTaskManager();

%testTaskMgr.addTask( "echo(\"Echo...\" );", 2);

%testTaskMgr.executeNextTask();
%testTaskMgr.executeNextTask();
%testTaskMgr.executeNextTask();

%testTaskMgr.delete();
```

Even though executeNextTask() was called three times, the output would only be:

```
Echo...
Echo...
```

Why? The task was told to execute twice, or to recycle N-1 times, where N was 2. Once this was done, the task was deleted from the queue. Note: A recycle value of 0 or 1 both mean 'run once', because (logically) a task can only be cycled a minimum of once.

Beyond finite repeats, a task can be made to recycle infinitely by doing this:

```
%testTaskMgr = newTaskManager();

%testTaskMgr.addTask( "echo(\"Echo A...\" );", -1);
%testTaskMgr.addTask( "echo(\"Echo B...\" );");

%testTaskMgr.executeNextTask();
```

```
%testTaskMgr.executeNextTask();
%testTaskMgr.executeNextTask();
%testTaskMgr.executeNextTask();

%testTaskMgr.delete();
```

The new output would be:

```
Echo A ...
Echo B ...
Echo A ...
Echo A ...
```

Preempting

In addition to repeating tasks, we can make some tasks preempt subsequent tasks:

```
%testTaskMgr = newTaskManager();

%testTaskMgr.addTask( "echo(\"Echo A...\" );", 2, true);
%testTaskMgr.addTask( "echo(\"Echo B...\" );", 2, false);

%testTaskMgr.executeNextTask();
%testTaskMgr.executeNextTask();
%testTaskMgr.executeNextTask();
%testTaskMgr.executeNextTask();
%testTaskMgr.executeNextTask();

%testTaskMgr.delete();
```

This code would produce:

```
Echo A ...
Echo A ...
Echo B ...
Echo B ...
```

What has happened is the first task was told to preempt subsequent tasks. To do this, the task manager executes the function and then places it at the front of the queue instead of the back.

Scheduled Preemption

The task manager allows us to add new tasks any time we wish. As we add tasks, we may wish to have them execute immediately instead of waiting their turn and working from back to front in the queue. Thus, this variation on the addTask() method is provided:

```
%testTaskMgr.addTaskFront( "echo(\"Echo A...\" );", 2, true);
```

Adding to the front of the task queue insures that this method will be executed next.

Return Values

SimpleTaskMgr can be used to drive decisions based on the values it returns when executing tasks. That is, if a task returns a value, the executeNextTask() method will return it:

```
function dummyTaskFunc(%val) {
    echo("dummyTaskFunc("@%val@")");
    return %val;
}

%testTaskMgr = newTaskManager();

%testTaskMgr.addTask( "dummyTaskFunc(" @ 10 @ ");", 2 );
%testTaskMgr.addTask( "dummyTaskFunc(" @ 20 @ ");", -1 );

%total = 0;
%total += %testTaskMgr.executeNextTask(); // +10
%total += %testTaskMgr.executeNextTask(); // +20
%total += %testTaskMgr.executeNextTask(); // +10
%total += %testTaskMgr.executeNextTask(); // +20
%total += %testTaskMgr.executeNextTask(); // +20

echo("Our total is ", %total);

%testTaskMgr.delete;
```

This produces:

```
Our total is 80
```

Late Evaluation

All tasks execute with the scope of the current task manager, thus we can pass variables to functions and methods that are scoped to the manager that is executing them. This gives us the ability to 'late evaluate' arguments. Used appropriately, this late execution feature can be used to create a very complex execution model. This sample of 'late evaluation' is taken directly from the SimpleTaskMgr validation code:

```
function TestObj::dummyTaskFunc( %this, %val) {
    echo("TestObj::dummyTaskFunc("@%val@")");
    return %val;
}
// Test target execution and ability to change task manager values
// and have them 'late evaluated'
function validateTaskMgr2() {
    %myObj = new scriptGroup(TestObj);

    %testTaskMgr = newTaskManager(%myObj);

    %testTaskMgr.addTask( "dummyTaskFunc( %this.val0 );" , 2);
    %testTaskMgr.addTask( "dummyTaskFunc( %this.val1 );" , 0);

    %testTaskMgr.val0 = 10;
    %testTaskMgr.val1 = 20;

    %total = 0;
    %total += %testTaskMgr.executeNextTask(); // +10
    %total += %testTaskMgr.executeNextTask(); // +20

    %testTaskMgr.val0 = 30;

    %total += %testTaskMgr.executeNextTask(); // +30
    %total += %testTaskMgr.executeNextTask(); // +0

    %myObj.delete();
    %testTaskMgr.delete();
}
```

Self-Execution

SimpleTaskMgr allows us to manually execute tasks and also provides the ability to schedule them. In other words, the task manager can be made to self-execute. When self-executing, the task manager will step over the queue till all tasks have been executed. Additionally, it can be told to execute tasks according to their own schedule or a fixed schedule:

```
%testTaskMgr = newTaskManager();

%testTaskMgr.setDefaultTaskDelay(2000);

%testTaskMgr.addTask( "echo(\"Task 0\" );" , 0, false, 500);
%testTaskMgr.addTask( "echo(\"Task 1\" );" , 0, false, 1000);

%testTaskMgr.selfExecuteTasks( true ); // Use Default Default Task Delay
```

In the above example, although the tasks have provided their own schedule times, these times will be overridden and they will be executed two seconds apart. Understand that delays come first, then execution, thus the execution would go like this:

```
// two second delay
Task 0
// two second delay
Task 1
```

Alternately, we could tell the task manager to use the tasks' times instead:

```
%testTaskMgr = newTaskManager();

%testTaskMgr.setDefaultTaskDelay(2000);

%testTaskMgr.addTask( "echo(\"Task 0\" );" , 0, false, 500);
%testTaskMgr.addTask( "echo(\"Task 1\" );" , 0, false, 1000);

%testTaskMgr.selfExecuteTasks( false ); // Ignore Default Task Delay
```

Now we get:

```
// half-second delay
Task 0
// one second delay
Task 1
```

In either case, if a delay value is set to -1, it means execute 'immediately':

```
%testTaskMgr = newTaskManager();

%testTaskMgr.addTask( "echo(\"Task 0\" );" , 0, false, 3000);
%testTaskMgr.addTask( "echo(\"Task 1\" );" , 0, false, -1);

%testTaskMgr.selfExecuteTasks( false );
```

Now we get:

```
// half-second delay
Task 0
Task 1
```

Stopping Self-Execution

In many cases we will need to stop the execution of our task manager and to re-start it at a later time. Therefore, a method is provided to stop the current execution:

```
%testTaskMgr.stopSelfExecution();
```

Stopping does not affect the contents of the task manager queue, nor does it stop the currently scheduled or executing task. The stop is applied after the next scheduled task.

Self-execution can be started again later with another call to 'selfExecuteTasks()'.

Deletion and Safety

As previously noted, SimpleTaskMgr is self-contained. It has been designed to handle deletion and to appropriately clean up. Subsequent tasks are canceled and the task list is cleaned up. Also, if the task manager is used with a target object and the target object is deleted or becomes invalid, the task manager will handle it and not attempt to execute against this object. Cycling will continue, if it is self-executing, but no work will be done.

It is best to create SimpleTaskMgr instances in objects' onAdd() methods and then to removed them in the objects' onRemove() methods.

Special Tokens

The task manager allows you to embed the following tokens in the task definitons:

Token	Purpose
TERMINATE#	Causes the task manager to self-terminate when it reaches this task.
LASTRET#	Replaces token with return value from last task.
LOCK#	Causes task manager to not store the return value for the current task, thereby retaining the last known return value.
STMT#	Treats the current task like a standalone statement or function call, regardless of whether this task manager has a 'target' object.
TASKMGR#	Replaces the token with the numeric ID of this task manager instance.
TASK#	Replaces the token with the numeric ID of the current task.
NULL#	This is an empty task. It is preferable to use delays on real tasks, but sometimes this is useful.

A.6.2. EGTGE Tasks Management Reference

EGTask:: Methods

A script object representing an individual task. This has two support function, only one of which should be used.

execute()

Execute this task. Normally, this is called by SimpleTaskMgr and should not normally be called directly by the user.

```
%myTask.execute();
```

setTaskDelay(***delay*** **)**

Set the automatic execution ***delay*** for this task. Delay is in milliseconds.

```
%myTask.setTaskDelay( 500 ); // Delay this task for 500 ms
```

SimpleTaskMgr Functions

newTaskManager([target])

Creates a new task manager with an optional ***target***, and returns ID.

```
%testTaskMgr = newTaskManager();
```

SimpleTaskMgr:: Methods

addTask(***task*** **, [** ***recycleCount*** **,** ***preempt*** **,** ***taskDelay*** **])**

Creates a new task at back of tasks queue and return ID of task.

- ***task*** – String specifying task to execute. Must be of a form acceptable to the eval() console function.
- ***recycleCount*** – Optional cycles to execute this task. Task is executed and pushed to back of task queue.
```
        0, 1  - Execute once.
        N > 1 - Execute N-1 times
        -1    - Execute forever.
```
-
- ***preempt*** – Boolean value specifying that recycled transactions should be pushed to front of task queue. Cannot be used with infinite ***recycleCount***.
- ***taskDelay*** – Number of milliseconds to wait prior to executing task when selfExecuting. -1 means execute immediately.

```
%testTaskMgr.addTask( "doit();" ) // Add task doit to back of queue
%testTaskMgr.addTask( "doit();" 2 ) // doit() twice
%testTaskMgr.addTask( "doit();" 2 , true) // doit() twice in a row
%testTaskMgr.addTask( "doit();" -1, false, 1000 ) // doit() once per second forever
```

addTaskFront(***task*****,** **[** ***recycleCount*****,** ***preempt*** **,** ***taskDelay*** **])**

Creates a new task at front of tasks queue and return ID of task. See addTask() for argument details.

```
%testTaskMgr.addTaskFront( "doit();" ) // Add task doit to front of queue
```

clearTarget()

Clear target for this manager. Once cleared, manager will assume tasks are not associated with an object and treat them like functions.

```
%testTaskMgr.clearTarget();
```

clearTasks()

Delete all tasks from queue without executing.

```
%testTaskMgr.clearTasks()
```

executeNextTask()

Executes next task in queue and returns value returned by task if any.

```
%retVal = %testTaskMgr.executeNextTask();
```

getTarget( )

Returns ID of current manager target object.

```
%curTarget = %testTaskMgr.getTarget();
```

selfExecuteTasks([***useDefaultDelay*** **])**

Causes task manager to execute tasks automatically (with delays if specified). If ***useDefaultDelay*** is true, all tasks will use the delay specified via 'setDefaultTaskDelay()' (see below), otherwise each task will use its own delay.

```
%testTaskMgr.selfExecuteTasks(); // Use task delays

%testTaskMgr.setDefaultTaskDelay( 1500 );
%testTaskMgr.selfExecuteTasks( true ); // Use default delay of 1.5 seconds
```

setDefaultTaskDelay(***delay*** **)**

Sets default delay for tasks when executed with over-ride delay. i.e. useDefaulDelay is true when calling 'selfExecuteTasks()' method.

```
%testTaskMgr.setDefaultTaskDelay( 1500 ); // Set default delay to 1.5 seconds

%testTaskMgr.setDefaultTaskDelay( -1 ); // Set default delay to immediate mode
```

```
setTarget( target )
 Set target to ID or Name supplied in target.
%testTaskMgr
```

```
stopSelfExecution( )
 Stop automatic execution of tasks.  Does not stop current scheduled or in-action task.

%testTaskMgr.stopSelfExecution();
```

A.6.3. Simple Inventory System (SimpleInventory)

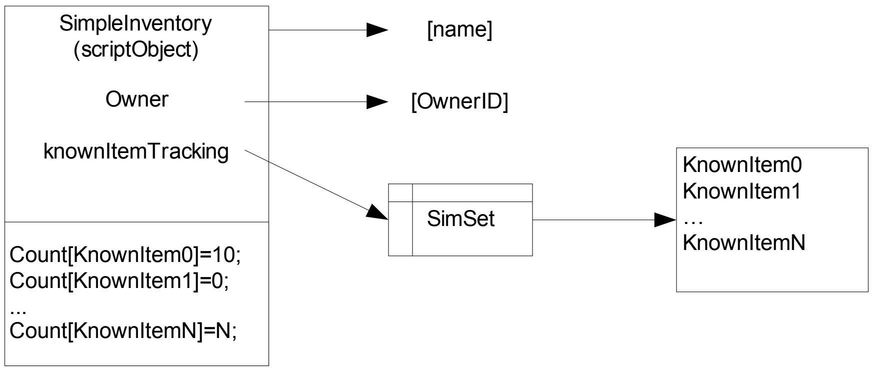
The Simple Inventory system provides the ability to create an inventory container and place it in any ShapeBase class. This container then scopes all the methods required to store and retrieve non-unique instances of Items. It also provides methods for constraining the inventory. i.e. One can specify max values for specific inventory items. Lastly, it provides a set of methods that are scoped to the ShapeBase/Data and Item/Data classes to handle all basic inventory interactions.

Described otherwise, SimpleInventory has the following attributes:

- It is script based and will work with the standard TGE kit.
- It is implemented with ScriptObjects and can be placed in any object or 'stand-alone'. In effect, this allows any object to have an inventory or inventories, further compartmenting and structuring game interactions.
- It is a generalized inventory system, designed to store non-unique items referenced by their datablock names.
- Items are stored and referenced by their datablock and can thus items with unique properties can be stored, but their unique-ness will be lost.
- An inventory can store any number of any type of datablock identified item.
- A max count limit can be set for any specific inventory item.
- All methods that operate on SimpleInventory are scoped under the SimpleInventory:: namespace.
- Inventory methods are provided for ShapeBaseData:: to enable a basic set of SimpleInventory interactions:
  - doPickup() – Pick up one instance of an object.
  - doThrow() – Throw or drop one instance of an object from inventory.
  - doUse() – Use an object from inventory.
- Inventory methods are provided for ItemData:: and Item:: classes to complete the inventory functionality.

A.6.4. SimpleInventory:: Structure

SimpleInventory Object



Inventoryable Items Datablock Template

```
datablock ItemData ( inventoryableItem )
{
    [InventoryItem  = alternateInventoryItemToStore;]
    [InventoryValue = 10;]

    // ...
};
```

- **InventoryItem** – Optional alternate item to store in inventory. Stores this items intead of collision item.
- **InventoryValue** – Optional inventory value for this item. By default, items are worth one inventory instance. This field can be used to increase that value. Values should only be positive.

A.6.5. Simple Inventory Console Functions

newSimpleInventory()

newSimpleInventory([%name])

Purpose:
Create a new SimpleInventory instance. Optionally give it the name **%name**.

Arguments:
%name – Handle to this inventory.

```
%Obj.myInventory = newSimpleInventory( backPack );
```

A.6.6. SimpleInventory:: Console Methods

addObject()	dumpContentsToString()	getInventoryCount()	getInventoryMaxCount()
listContents()	removeObject()	setInventoryCount()	setInventoryMaxCount()
setOwner()	verifyArgs()		

addObject(%theInventory , %objectName [, %numobjects])

Purpose:
Add one %objectName object to the inventory. Returns number of objects successfully added.

Arguments:
%theInventory – Handle to this inventory.
%objectName – Datablock ID for this object.
%numObjects – Number objects to add.

dumpContentsToString(%theInventory)

Purpose:
Returns semi-colon (;) separated list of current inventory contents in the form:

ObjName0;Count0;...ObjNameN;CountN

Arguments:
%theInventory – Handle to this inventory.

getInventoryCount(%theInventory , %objectName)

Purpose:
Get total number of %objectName objects in the inventory. Returns 0 if none found.

Arguments:
%theInventory – Handle to this inventory.
%objectName – Datablock ID for this object.

getInventoryMaxCount(%theInventory , %objectName)

Purpose:
Get max number of %objectName objects allowed in the inventory.

Arguments:
%theInventory – Handle to this inventory.
%objectName – Datablock ID for this object.

listContents(%theInventory)

Purpose:
Prints a list of the inventory's contents to the console.

Arguments:
%theInventory - Handle to this inventory.

removeObject(%theInventory , %objectName [, %numobjects])

Purpose:
Remove one %objectName from the inventory. Returns number of objects successfully removed.

Arguments:
%theInventory - Handle to this inventory.
%objectName - Datablock ID for this object.
%numObjects - Number objects to remove.

setInventoryCount(%theInventory , %objectName , %numObjects)

Purpose:
Set total number of %objectName objects in the inventory. Returns number of objects succesfully set.

Arguments:
%theInventory - Handle to this inventory.
%objectName - Datablock ID for this object.
%numObjects - Number objects to set.

setInventoryMaxCount(%theInventory , %objectName , %maxCount)

Purpose:
Sets maximum number of %objectName objects allowed in the inventory.

Arguments:
%theInventory - Handle to this inventory.
%objectName - Datablock ID for this object.
%maxCount - Max objects of this type allowed. (Can be "", 0, or N > 0)

setOwner(%theInventory , %ownerObj)

Purpose:
Assigns and 'owner' to this inventory.

Arguments:
%theInventory - Handle to this inventory.
%ownerObj - Object that contains/owns this inventory.

verifyArgs(%theInventory , %objectName)

Purpose:
Verifies that %objectName is in fact an object and returns the stringized name of said object.

Arguments:
%theInventory - Handle to this inventory.
%objectName - Datablock ID for this object.

A.6.7. ShapeBaseData:: Inventory Methods

doPickup()	doThrow()	throwObject()	doUse()

doPickup(%ownerDB , %ownerObj , %pickupObj)

Purpose:
Do a pickup of %pickupObj.

Returns:
- 0 – Fail
- N > 0 – Success. N == number of objects added (always 1 for SimpleInventory).

Arguments:
%ownerDB – Datablock ID or string for this owner.
%ownerObj – Handle to this owner.
%pickupObj – Object to pick up.

Notes:
- Assumes inventory field is named: myInventory

doThrow(%ownerDB , %ownerObj , %throwDB)

Purpose:
Do a throw of %throwDB. If the throw is successful, call %throwObj (returned by onThrow().schedulePop() to remove the object in $Item::PopTime milliseconds.

Returns:
- 0 – Failed throw.
- 1 – Succesful throw.

Arguments:
%ownerDB – Datablock ID or string for this owner.
%ownerObj – Handle to this owner.
%throwDB – Datablock ID or string for throw object.

Notes:
- Assumes inventory field is named: myInventory

throwObject(%ownerDB , %ownerObj , %throwObj)

Purpose:
Execute the actual 'throwing' of the object. This function simply applies a mass specific impulse to the throw item along upward-diagonal vector calculated from the owner's eye vector.

Arguments:
%ownerDB – Datablock ID or string for this owner.
%ownerObj – Handle to this owner.
%throwObj – Object to throw.

doUse(%ownerDB , %ownerObj , %useDB)

Purpose:
Attempt to use execute the onUse() method fo %useDB.

Returns:
- 0 – Fail
- N > 0 – Success. N == number of objects added (always 1 for SimpleInventory).

Arguments:
%ownerDB – Datablock ID or string for this owner.
%ownerObj – Handle to this owner.
%useDB– Object to use.

Notes:
- Assumes inventory field is named: myInventory
- This function may seem to fulfill no real purpose, but it actually an ideal place to put additional use functionality for different ShapeBase derived class types.

A.6.8. ItemData:: Inventory Methods

onInventory()	onPickup()	onThrow()	onUse()

onInventory(%inventoryObj , %ownerObj, %amount)

Purpose:
This method is called for all items when the value (count) of this item is changed (either added or removed) in inventory.

Returns:
- 0 – Fail
- N > 0 – Success. N == number of objects added (always 1 for SimpleInventory).

Arguments:
%inventoryObj – Datablock ID or string for this object.
%ownerObj – Object that owns inventory.
%amount – Non-zero change value (either positive for add, or negative for remove)

Notes:
- Assumes inventory field is named: myInventory

onPickup(%pickupDB , %pickupObj, %ownerObj)

Purpose:
Attempt to place %pickupDB in %ownerObj inventory. If the add is successful, call %pickupObj.respawn() to temporarily hide the object. The object will reappear in $Item::RespawnTime milliseconds

Returns:
- 0 – Fail
- N > 0 – Success. N == number of objects added (always 1 for SimpleInventory).

Arguments:
%pickupDB – Datablock ID or string for this object.
%pickupObj– Handle to this item object.
%ownerObj – Agent calling onPickup().

Notes:
- Assumes inventory field is named: myInventory

onThrow(%throwDB , %ownerObj)

Purpose:
Attempt to remove %throwDB from %ownerObj inventory. If successful, create a new instance of type % throwDB and return the handle %throwObj.

Returns:
- 0 – Item not found in inventory, or failed to create instance.
- %throwObj – New object handle.

Arguments:
%throwDB – Datablock ID or string for this object.
%ownerObj – Agent calling onThrow().

Notes:
- Assumes inventory field is named: myInventory

onUse(%useDB , %ownerObj)

Purpose:
Attempt to remove %useDB from %owneObj inventory. If successful, 'do something' with the object.

Returns:
- false – Item not found in inventory.
- true – Successfully 'used' object.

Arguments:
%ItemDB – Datablock ID for this object.
%Owner – Agent calling onUse().

A.6.9. Item:: Inventory Globals

```
$Item::PopTime
 Time afer pickup till items pop (auto-delete).  Default == 5000 milliseconds.
```

```
$Item::RespawnTime
 Time afer pickup till items respawn (un-hide).  Default == 5000 milliseconds.
```

A.6.10. Item:: Inventory Helper Methods

respawn()	schedulePop()

```
respawn( %Item )
 Purpose:
 Fades out and hides %Item.  Then schedules an un-hide and fade-in to occur in
$Item::RespawnTime milliseconds.

 Arguments:
 %Item - Handle to object to operate on.
```

```
schedulePop( %Item )
 Purpose:
 Schedules a fade-out and delete() on %Item in $Item::PopTime millisconds.

 Arguments:
 %Item - Handle to object to operate on.
```

A.6.11. GPGT Utilities

This section documents various utility scripts that have been included with the book for your use. These scripts were created to ease some of my coding tasks and I thought they would also be useful to you, the reader. Enjoy.

A.6.11.1. String Utilities

```
randomizeWords( %words , %iterations )
 Purpose:
 Randomizes the ordering of all words found in the %words string, returning a new string
containing the result.  Randomization can be improved by increasing the %iterations
setting.

 Arguments:
      %words - A string containing word to randomize.
 %iterations - The number of passes to make while randomizing the list.
```

```
%wordList = "This is a test";

%randWordList = randomizeWords( %wordList , 10 );
```

swapWords(%words, %first, %second)

```
 Purpose:
 Swaps two words, found in %words and located at indexes %first and %second.  Returns a
string containing the contents of %words, with the two strings re-ordered.  If either
index is out of bounds, the original string is returned.

 Arguments:
  %words - A string containing words, two of which will be swapped.
  %first - An integer value indicating the index of the first word.
 %second - An integer value indicating the index of the second word.
```

```
%wordList = "This is a test";

%swapWordList = swapWords( %wordList , 0, 2 );

echo( %swapWordList ); // Produces "a is This test"
```

A.6.11.2. SimSet Utilities

These utilities are used to manipulated SimSet, SimGroup, and ScriptGroup objects.

simSet::copyToSet(%theSet, %destSet)

```
 Purpose:
 Copies the contents of %theSet to %destSet.  There is no return value.

 This can also be used to move the contents of one SimGroup or ScriptGroup to another.

 Arguments:
  %theSet - SimSet to copy contents from.
 %destSet - SimSet to copy contents to.
```

```
new SimGroup(source);
new SimGroup(dest);

source.add( new SimObject(A) );
source.add( new SimObject(B) );

source.copyToSet( dest );

source.add( new SimObject(C) );

source.listObjects(); // Lists only  C

dest.listObjects(); // Lists A and B
```

simSet::deleteSet(%theSet, %selfDestruct)

```
 Purpose:
 Iteratively deletes the contents of the set %theSet.  Optionally, if %selfDestruct is set
to true, the object will then self-delete.

 This is not the same as clear(), but rather it actually deletes every object stored in
the simSet.

 Arguments:
       %theSet - The SimSet from which to remove and delete all contents.
 %selfDestruct - A boolean value, when set to true, indicating that the
                 SimSet should be deleted after the contents have been removed.
```

```
%myTask.execute();
```

simSet::forEach(%theSet, %function, [%isMethod , [%printEval]])

```
 Purpose:
 This method allows us to iteratively call a function or a method on every entry in the
SimSet.

 Arguments:
    %theSet - The SimSet whose contents will be called against.
  %function - The unadorned name of the function or method.
  %isMethod - A boolean value, when true, indicating %function is a method,
              otherwise %function is treated as a function.
 %printEval - A debug feature that will print the strings being fed to eval().
              If you are having trouble with this method, set this to true
              to see what scripts are actually being constructed and executed.
```

```
function test_func( %obj )
{
    echo("test_func got %obj == ", %obj.getname() );
};

function SimSet::test_method( %obj )
{
    echo("test_method got %obj == ", %obj.getname() );
};

new SimSet( testSet );

testSet.add( new SimObject( A ) );
testSet.add( new SimObject( B ) );
testSet.add( new SimObject( C ) );


// There are three ways to call the above function and method
//

// 1
testSet.forEach( test_func ); // call test_func as method
```

```
// 2
testSet.forEach( test_method , 1 ); // call test_method as method

// 3
testSet.forEach( SimSet::test_method  );  // treat test_method like function
```

simSet::forEachStmt(%theSet, %statement, %token, %printEval)

```
 Purpose:
 This method allows us to iteratively call a statement using each object in the SimSet.

 This will call a series of statements, each operating on one of the objects in the set at
a time.

 Arguments:
   %theSet  - The SimSet whose contents will be called against.
 %statement - Any legal TorqueScript statement (with closing semi-colon (;)).
     %token - A token name embedded in %statement which will be replaced with
              the ID of the object that is currently being operated on.
              The token can be used multiple times in the same statement.
 %printEval - A debug feature that will print the strings being fed to eval().
              If you are having trouble with this method, set this to true
              to see what scripts are actually being constructed and executed.
```

```
$x = new SimSet();

$x.add( new SimObject( A ) );
$x.add( new SimObject( B ) );
$x.add( new SimObject( C ) );

$x.forEachStmt("echo( tok.getName(), \" has ID == \", tok.getID() );" , tok );

// Above code call would print something like:
// A has ID == 123
// B has ID == 124
// C has ID == 125
```

simSet::getRandomObject(%theSet)

Purpose:
Returns the ID of an object from **%theSet**, selected at random. Also, for any set with two or more objects stored in it, this method is guaranteed to never return the same value twice in a row.

Warning:
This method adds a field to all sets that call it named: *_lastRandomObject*. This field is used to track the last object returned and allows the method to protect against repeats.

Arguments:
%theSet – The SimSet to get our random selection from.

```
$x = new SimSet();

$x.add( new SimObject( A ) );
$x.add( new SimObject( B ) );
$x.add( new SimObject( C ) );

echo( $x.getRandomObject );
echo( $x.getRandomObject );
echo( $x.getRandomObject );
echo( $x.getRandomObject );

// Above echo calls might print something like:
// B
// A
// C
// A
```

A.6.11.3. Array Object

The purpose of these utilities is to allow the creation of array objects that allow easy sorting and passing of arrays. Normal Torque arrays cannot be passed as arguments to functions or methods, and sorting them requires that you write code. The array object utilities now take care of this for you. With the provided utility methods, you may create a scriptObject (or scriptGroup) using any of the following syntaxes:

```
%aryObj = new ScriptObject( arrayObject );

// OR

%aryObj = new ScriptObject(  )
{
   class = "arrayObject";
};

// OR

%aryObj = new ScriptObject(  )
{
   superClass = "arrayObject";
};
```

Subsequently, you may use this object as an array as follows:

```
// Add four entries to the array object

%aryObj.addEntry( "This" );
%aryObj.addEntry( "Is" );
%aryObj.addEntry( "a" );
%aryObj.addEntry( "test" );

// Get the count of objects in this array
echo( "My array object has ", %aryObj.getCount() , " entries.");

// Sort the array (increasing sort only)
%aryObj.sort();

// Pass the array to a function for printing
// Just demonstrating ability to pass the array
function dumpArray( %aryObject )
{
    for( %count = 0; %count < %aryObject.getCount(); %count ++ )
    {
        echo("Entry(", %count , ") == ", %arrayObject.getEntry( %count ) );
    }
}

dumpArray( %aryObj );
```

The result of the above dumpArray() call will be:

```
Entry(0) == a
Entry(1) == Is
Entry(2) == test
Entry(3) == This
```

arrayObject::addEntry(%Obj , %entry)

```
Purpose:
Adds a new entry into the array.

Arguments:
  %Obj - The array Object.
%entry - New value to add to array.
```

arrayObject::getCount(%Obj)

```
Purpose:
Returns the number of entries in the array.

Arguments:
%Obj - The array Object.
```

arrayObject::getEntry(%Obj , %index)

```
Purpose:
Returns the entry in this array at index %index, or "" if no entry is found.

Arguments:
  %Obj - The array Object.
%index - Index to entry to retrieve.
```

arrayObject::sort(%Obj [, %Decreasing])

Purpose:
Sorts the contents of the array in increasing order if **%Decreasing** is set to false, or not specified. If **%Decreasing** is set to true, the array is sorted in decreasing order.

Warning:
This sort uses a lexicographic comparison, meaning the entries <5 5000 a b c> would be sorted in these orders:

```
Increasing Order -> < 5 5000 a b c >

Decreasing Order -> < c b a 5000 5 >

Arguments:
       %Obj - The array Object.
%Decreasing - Optional boolean value. Setting this to true sorts in
              decreasing ordering.
```

A.6.11.1. Miscellaneous Utilities

CalculateObjectDropPosition(%oldPosition , %offsetVector)

Purpose:
This function will calculate a drop point without moving the object and returns the calculated drop position.

The function casts a ray from %oldPosition downward and calculates a drop position based on the first object that is hit. This ray cast will hit ALL game objects that have a collision mesh.

```
Final positions will be:
                        "initially calculated drop" + %offsetVector

Arguments:
      %object - The object to be dropped.
%offsetVector - An offset to adjust the drop by.
```

DropObject(%object , %offsetVector)

Purpose:
This function will drop an object to the ground, or on top of the first object found with a valid collision box below the dropping object.

The dropping object starts its 'drop' at the position it was created at and drop from there.

Note 1:
To calculate a starting position, use CalculateObjectDropPosition().

Note 2:
This function will not move TSStatic() objects. Their positions must be set once and only once, upon construction. Use CalculateObjectDropPosition() instead.

Arguments:
```
      %object - The object to be dropped.
%offsetVector - An offset to adjust the drop by.
```

DropObjectFromMarker(%object , %marker , %offsetVector)

Purpose:
This is the same as DropObject, but it drops the object from the position of another world object whose handle is passed in **%marker**.

Arguments:
```
      %object - The object to be dropped.
      %marker - A valid marker (object) to be used as a starting point.
%offsetVector - An offset to adjust the drop by.
```

getLeftVector(%vec)

Purpose:
Returns the pre-normalized right-hand cross-product of %vec and the world up-vector. i.e., This operation is performed: (| vec | X "0 0 1")

Arguments:
%vec - The vector to find a left vector for.

```
echo( getLeftVector(" 100 50 75 " )  ); // prints:  < 0.447 -0.894 0 >
```

getRightVector(%vec)

Purpose:
Returns the pre-normalized left-hand cross-product of %vec and the world up-vector. i.e., This operation is performed: -(| vec | X "0 0 1")

Arguments:
%vec - The vector to find a left vector for.

```
echo( getRightVector(" 100 50 75 " )  ); // prints:  < -0.447 0.894 0 >
```

GuiMLTextCtrl::fillFromFile(%theControl , %textFile [, %clear])

```
 Purpose:
 This handy helper adds the contents of any file to any GUIMLTextCtrl object.  You may
optionally clear the current contents or append to them.

 Arguments:
 %theControl - Handle to the GUIMLTextCtrl.
   %textFile - Path to the file to be loaded.
      %clear - Optional boolean value.  If set to true, the control is cleared
               before loading occurs.
```

```
// Place the contents of soHandy.txt into myGUIMLTextCtrl, and clear it first
myGUIMLTextCtrl.fillFromFile( "~/soHandy.txt", true );
```

sceneObject::getCompassPoints(%Obj, %Offset)

 Purpose:
 Returns a SimSet containing scriptObjects, each having a field named 'position' containing the location of a compass point about the shape.

 Compass points are in a circle about the shape, where the circle lies on the outer edges of the shape's world box. The circle's radius can be increased by specifying an **%Offset**, where **%Offset** will be added to the initial radius.

```
 Arguments:
    %Obj - The sceneObject to find eight compass points for.
 %Offset - A floating-point value by which to increase the radius of the base
           circle on which the compass points lie.
```

```
// Place eight spawn spheres around an object.

function scriptObject::placeMarker( %Obj )
{
   %obj = new SpawnSphere()
   {
      dataBlock = %data;
      position  = %Obj.position;
   };

   MissionCleanup.add( %obj );
}

%compassSet = myObject.getCompassPoints();

%compassSet.forEach( placeMarker );

%compassSet.deleteSet( true );  // Destroy the set and all markers in it.
```

SceneObject::getLeftVector(%Obj)

Purpose:
This method passes the forward vector of **%Obj** to getLeftVector and returns the result.

Arguments:
%Obj – The object to get the left vector for.

```
echo( getObjLeftVector( %player ) );
```

SceneObject::getRightVector(%Obj)

Purpose:
This method passes the forward vector of **%Obj** to getRightVector and returns the result.

Arguments:
%Obj – The object to get the left vector for.

```
echo( getObjRightVector( %player ) );
```