v020327

IAGEScript Developer Reference

27th March, 2002

Rawson-Tetley Data Systems
www.robin.rawsontetley.btinternet.co.uk
mailto:iage@btinternet.com

IAGE Game File Format Specification

Header

The header of the IAGE file contains two lines, the first of which contains the text “IAGE 2”.

The second will be a 0 or 1 indicator, set to 0 if the file is unencrypted, or 1 if the file is encrypted.

Sections

The IAGE file is separated into sections, starting with the section name in capital letters, containing any number of properties. A line containing the word BREAK must be used to separate collections of objects. A line containing the word END will terminate the section.

eg:

LOCATIONS
(properties)
BREAK
(properties)
END

Properties

IAGE Properties are stored as name/value pairs under the appropriate section. The equals sign is taken to be the separator of the name/value pair and the Windows carriage return/linefeed (13 + 10) is taken to be the line terminator.

eg. Name = Smith.

Properties representing IAGEScript

Where properties represent a collection of IAGEScript code, the property name will be reiterated for each line - eg:

OnInput = ‘ IAGE Code
OnInput = ‘ Written by Rob
OnInput =
OnInput = currentplayer.print Some^text.

Comments

Any line which does not have an equals sign or represent a start marker is ignored and treated as a comment.

IAGEScript Reference

Variables

Variables are ways of temporarily storing a value. I say temporarily because the only forms of variable IAGE supports are local variables. Variables declared in a script cannot be accessed from outside that script. You must use the flag objects (and indeed other IAGE object properties) to persist variable data.

Variables are declared with the VAR command. All variables in IAGE are variants, which can hold numbers, strings, boolean (true/false) values and arrays.

Variables can be initialised when they are declared also, Eg:

var x = 1

Would create a new variable “x” with a value of 1.

var orc

Would create a new variable “orc” with an uninitialised value. Uninitialised variables by default contain an empty string, it is better to initialise variables to zero.

Arrays

All variables in IAGE are of variant type - they can hold strings or numbers or arrays. To manipulate arrays, you use the inbuilt array object. Arrays can be persisted in any IAGE variant property (flags, get/setvalue). The array object supports the following methods:

new <name> <itemlist>

This method creates a new array and gives it the name you specify and the contents you list.
Eg: array.new MyArray item1 item2 item3 item4 item5
Would create an array called “MyArray” containing the 5 values listed after it.

add <name> <item>

This method appends a new item to the end of the array.
Eg: array.add MyArray item6
Would add a 6th item to the array created in the example before.

get <name> <itemno>, set <name> <itemno> <value>

Retrieves the numbered item value from the array. Note that because it uses multiple arguments you cannot use array.get in expressions. You need to declare a variable to hold the return value and then use that in your expression. Set is used to explicitly set an array value (it must already exist to do this).

Eg: var myvar = array.get MyArray 3
Returns item 3 of the array into variable “myvar”.

remove <name> <itemno>

Deletes the specified item from the array. If there are any higher entries in the array, they will be shuffled down.

getcount(<name>)

Returns the number of elements in the array. You are safe to use this method in expressions.

Syntax - Referring to Objects and Collections

IAGE supports two types of objects - collection (or multi-instance) objects and single instance objects.

Single Instance Objects

You refer to single-instance objects to retrieve/set properties and call methods by using the object name, followed by a dot (.) and then the name of the method or property you wish to access.

Eg: To store the second noun a player entered in a variable called “x”, I would use the following code:

var x = input.noun2
To redisplay the current location to a player, you need to call the “displaycurrentlocation” method of the game object, like this:

game.displaycurrentlocation
Unlike most languages, when you make a method call which has no arguments in IAGE, you do not supply the brackets.

Multi Instance Objects

Multi Instance Objects work just the same as Single Instance Objects, the only difference is that you need to supply the ID of the object you wish to work with so IAGE knows which object within the collection you are talking about.

Eg: If I wanted to display message with ID 53 to the current user:

message(53).show
Or perhaps, I could move item 6 to a different location:

item(6).currentlocation = 3

Since items used named arguments, this would more typically be something like mailbox.currentlocation = 3

Arguments

Sometimes, a method call requires arguments. If this is the case, simply include an extra set of brackets after the method call containing the arguments.

Eg: To determine whether user boolean “islaser” exists in item 26:

if (item(26).getuserboolean(islaser) = true) then

Maths:

Syntax:

All maths and expressions should be surrounded by pipes (|) to denote an expression. Only simple plus, minus, divide and multiply mathematics are supported, although there is no limit on the complexity of the numbers involved.

String concatenation is also supported via the ampersand character.

In IAGE, mathematic processing is sequential. It literally starts with the first number, applies the second number via the operand, then the third, then so on. If you take the example: 1 - 5 * 4, you will get an answer of –16. This is because IAGE will evaluate the expression to 1-5 = -4 * 4 = -16.

If you performed this operation in most other languages, you would get the answer of –19 because it would be evaluated to 1 – (5 x 4). This is because multiplications take precedence over addition and subtraction.

Symbols:

+	Plus
-	Minus
/	Divide
*	Multiply
&	Concatenate
.	Decimal Point

Functions:

Random	A Random number between 0 and 1.
MakeInteger	Always use with + to make the current result into an integer. Integers always round down.
Space	Inserts a space

Examples:
Flags(1).Value = |1 - 5 / 16 * 28 + Flags(298).Value|
(translates to 1 minus 5 divided by 16 multiplied by 28 add the contents of Flag 298)

Message(1000).Text = |Flags(298).Value & Message(1).Text|
(translates to the contents of Flags 298 in string form with the text of message 1 on the end)

Flags(2).Value = |Random * 1000 + 1 + MakeInteger|
(translates to a number between 1 and 1000 which is then rounded down to be made a whole number)

Code flow commands:

Goto <number or label>

Sends execution to the specified line number (see line numbering and labels above). Do not include the semi-colon at the end of the line number or label using GoTo.

End

Terminates execution of the current code module. Useful when a condition is fulfilled and you do not wish to execute the remainder of the code.

CancelEvent

Used to tell the system not to perform an action. For example, when the player quits the game, the Game.OnQuit event runs. If a CancelEvent instruction is executed in the code, the player does not quit..

Conditional Commands

If (condition) .. Then .. Else .. EndIf

If conditional blocks should follow the format of:

```
If ( condition ) Then
    Execute code
Else
    Execute code
EndIf
```

If there is no else clause, it can be omitted, but you MUST terminate with an EndIf. Note that line breaks MUST be used in the correct places. The If always appears on it's own line, as does the else and the EndIf.

Ifs can be quite safely nested to an unlimited depth.

Condition Syntax

Conditions should be contained by brackets with a space at either end. Operators should be outside the brackets and only one expression at a time should be inside the brackets.

Eg:

```
If ( flag(12).Value = true ) AND ( flag(14).value = false ) then
If ( currentplayer.cansee = true ) OR ( location(currentplayer.currentlocation).isdark = false ) then
```

Boolean Comparison

Boolean AND and OR is allowed in if (and while) statements. These comparisons must be outside and separating each bracketed expression.

Shorthand IF statements

IAGE allows two forms of shorthand IF statements, represented by the semi-colon (;) and the hash (#).

Semi-colon<verb text> is equivalent to "if (input.verb = x) then" where x is the ID of the verb text you enter. This means that you can use the English words instead of having to memorise the verb IDs. It also means that regardless of the verb text you enter, any of that verb's synonyms will evaluate to true.

Eg:

```
;smell
endif
```
is the same as:

```
if ( input.verb = 8 ) then
endif
```

It’s just much more legible!

You can also use multiple verb texts, broken with a comma. The comma represents an OR comparison.

So:

;look,smell,listen
endif
Would evaluate to true if any of those verbs (or their synonyms) was the verb the player entered.

Finally, the hash symbol is functionally identical, but instead of checking the first verb, it checks the first adverb - highly useful for checking for directions.

Eg:

#north,south,east
endif

Would evaluate to true if the player entered north, south or east (or any of their synonyms).

Loops

At present IAGE only supports while/endwhile loops.

Start a while loop with the syntax “while <condition> then” and terminate the loop with “endwhile”.

The loop will execute whilstever the condition is evaluates to true.

The syntax for the condition is identical to the if statement.

Eg:

```
var x = 1
while ( x < |currentplayer.count + 1| ) then
   currentplayer.print playerarray(x).name
   x = |x + 1|
endwhile
```

This would create a loop which enumerated every player in the game and displayed their name to the current player.

Media Instructions

These commands are used to access IAGE's multimedia functionality.

ShowPicture <name>

Displays the picture with <name>. You may use either a fully qualified URL, or one relative to the media path set in the game preferences.

NB: The ShowPicture functionality was removed in a much earlier version of IAGE, however all references to it still exist. It will be re-added in the near future.

Functions

IAGE has some additional, handy functions which are very useful when writing a game.

Note that function syntax is slightly different. Because they behave like standalone objects, you do not wrap brackets around any arguments. This means that they cannot be used in expressions directly. However, you can create a variable to catch the returnvalue and then use that in an expression.

GetItemFromNoun

The GetItemFromNoun function does just that, you supply an expression which evaluates to a Noun ID and it will return the ID of the item which uses this Noun ID as a base.

Eg:

```
var imid = getitemfromnoun input.noun
currentplayer.print item(imid).name
```

Would display the name of the first item a player referred to.

Ask

The Ask function accepts an expression which equates to some text. The text is then output to the current player and their game is frozen. The next response they make will be the response to the ask question (and cannot be subjected to dictionary matching, etc.).

The player response will be returned as a string.

Eg:

```
var result = ask “Do you like burgers?”
if ( result = n ) or ( result = no ) then
   currentplayer.print “What the hell is the matter with you?”
endif
```

Would ask the player if they liked burgers and insult them if they didn’t.

OutputContentsOf

The OutputContentsOf function accepts two expressions which equate to a playerindex and a locationid

The contents of that location will be transmitted to the player at the index specified.

Eg:

```
outputcontentsof currentplayer.index westofhouse.id
```

Would show a list of all the items in location named “westofhouse” to the current player.

PrintAllIn

Accepts two expressions, one representing a location ID and one representing the message to be output.

Eg:

printallin westofhouse.id message(58).text

Would output the text of message 58 to everyone in location “westofhouse”.

PrintAllExcept

Accepts two expressions, one representing a player index and one representing the message to be output.

Eg:

printallexcept currentplayer.index “Yahboosucks!”

Would output the message “Yahboosucks!” to everyone but the current player.

PrintAllInExcept

Accepts three expressions, a location ID, a player index and a message.

Eg:

printallinexcept 4 currentplayer.index “Not you current player!”

Would ouput “Not you current player!” to everyone in location 4 but the current player.

AddVerb, AddAdverb, AddNoun

These three commands allow you to dynamically add to the dictionary in a game. You can add nouns directly in an item’s initialise event, or do dictionary entries en-masse in the game’s initialise event. You can even add them while a game is running.

Each command accepts two paramters, an ID number and the text of the dictionary entry. Note that you are not permitted to use expressions here - you must use number and string literals.

Eg:

AddVerb 601 Info
AddNoun 1001 Spoon
AddAdverb 54 with

Location ID Constants:

By assigning the location ID of various IAGE properties to one of the following range of values, objects can be contained, carried, on surfaces, in locations, or in limbo.

100,000 + Player Index	Carried by Player
200,000 + Object ID	Contained by Object ID
300,000 + NPC ID	Carried by NPC ID
400,000 + Object ID	On Surface of Object ID
Between 0 and 99,999	Location ID
0	Limbo
-1	Randomly Generated Location ID

Although this is technically how IAGE works underneath, it is highly recommended that you use the containerlocation and surfacelocation properties of objects as these constants may change. Also note, the keyword “random” can be used anywhere a location name is expected.

Code On the Fly:

IAGE allows you to generate its core objects (NPCs, Items and Locations) entirely on the fly in code while your game is running by using the following commands.

AddNPC, AddLocation, AddItem

All of these accept one argument - an ID value for the created item. Eg:

AddLocation 153
Location(153).Name = “West Of House”
Location(153).Description = “You are standing in a field, west of a white house.”

Would create a new location with ID 153 called “West Of House”.

RemoveNPC, RemoveLocation, RemoveItem

These also accept an ID argument and remove that object from the game.

Object Specific

All game objects implement the following methods to allow their code to be manipulated by IAGE code that is already running:

<object>.CodeClear <Event>

Wipes all code from that particular event for that object.

<object>.CodeAddLine <Event> <String>

Adds the code in <String> to the event named on the object. Eg:

leaflet.CodeAddLine “OnAction” “;examine”
leaflet.CodeAddLine “OnAction” “currentplayer.print “Yo!”
leaflet.CodeAddLine “OnAction” “endif”

Would override the leaflet’s examine.

<object>.FireInLineEvent <Event> <Inline Event>

Runs the procedure named <Inline Event>, stored in the event named for the object. NOTE: You can actually make up your own inline events - in effect creating your own object methods.

Character Object (Multi-Instance)

The Character object collection contains information about all the NPCs (Non-Player Characters) in a game.

Properties	Methods	Events
Count	SetValue	OnTimer
ID	GetValue	OnTalk
Name	CodeClear	OnAction
CurrentLocation	CodeAddLine	
HitPoints	FireInlineEvent	Inline (OnAction)
DamageIndicator		
Money		Before_Display
AutoAttack		Before_DisplayInitial
AttackWhenAttacked		Initialise
AttackingWho		Each_Turn
Description		
DefaultExamine		
NounID		
TimerInterval		
TimerIndex		
TimeToNextRun		
MovedFromOriginalLocation		
AIMode		
FollowPlayerIndex		

Count

Returns the number of NPC objects in the game. As with all object/collections in IAGE, you do not need to specify an individual item when requesting a count, you can omit the bracketed part completely (eg. character.count).

ID

The ID of the NPC.

Name

The name of this NPC. This is what will be displayed to players once the NPC has it's "MovedFromOriginal-Location" flag set to true.

CurrentLocation

The ID of the location this NPC is currently in.

HitPoints
(If using IAGE combat) The number of hit points this NPC has remaining before it dies.

DamageIndicator
The maximum amount of damage this NPC can cause during combat.

Money
The amount of money this NPC is carrying.

AutoAttack
If set to true, this NPC will automatically attack players as soon as they come into the same location as the NPC.

AttackWhenAttacked
If set to true, this NPC will automatically attack players when they attack the NPC.

AttackingWho
If the NPC is currently attacking someone, this is the Index property of the player being attacked.

Description
The text shown instead of the NPC name when MovedFromOriginalLocation is false.

DefaultExamine
The default text displayed when the NPC is examined.

NounID
The ID of a group of nouns referring to this NPC.

TimerInterval
The number of milliseconds before the OnTimer event is run.

TimerIndex
The number of times since the server was started that the OnTimer event has been run for this NPC.

TimeToNextRun
Arbitrary value in milliseconds determining the time the OnTimer event will run. Do not use this property for anything as it's value is highly volatile.

MovedFromOriginalLocation
Determines whether the name or description is shown when the NPC is displayed. If this is true, the name is shown.

FollowPlayerIndex
If using AIMode = 2, the index of the player this NPC is following.

AIMode
One of three different settings representing actions IAGE can make the NPC do for you. Set this property to one of the following values:

0 - Dormant (NPC does nothing but execute events)
1 - Follow Player (Set followplayerindex and NPC will follow them around)
2 - Random (NPC moves randomly around the map)
3 - Attacking (NPC attacks the player who's index is specified in the AttackingWho property)

SetValue
Accepts a parameter name and allows you to assign a named value to this item. If a value exists with that name, it is updated, otherwise it is created.

Eg:

thief.setvalue(doorcount) = 58

GetValue
Accepts a parameter name and returns the value for this named argument.

Eg:

currentplayer.print mat.getvalue(doorcount)

if (mat.getvalue(doorcount) = 58) then
  currentplayer.print "It's 58!"
endif

OnTalk
The OnTalk event fires when a player attempts to talk to the NPC. Inbuilt conversation is handled by a player entering <npcname> followed by a comma, and then the actual conversation. Eg. "Gandalf, give me the cheese". This event is also fired when a player uses "Say" in a location with an npc. You can use the input.contains method to parse the text.

Note that when you are running script in the OnTalk event, IAGE shuffles the Nouns down one place for you (ie. You already know the first noun because it's the NPC, so IAGE discards it and shuffles Nouns 2 and 3 down to become 1 and 2).

OnTimer
The OnTimer event fires every time the duration of the TimerInterval property elapses. The TimerIndex is also incremented every time this event is run.

Use the OnTimer event to implement AI. Eg. You could switch on random AI mode and then have the NPC check players in it's current location on the timer and "relieve them of a few valuables" like the thief in Zork, or perhaps picking up and dropping objects at random to make the player's job harder. It's entirely upto you.

Note that the timers run in real time - ie. They don't wait for player responses. This is because players don't wait for each other's responses in multiplayer games. If you are writing a single player game and you do not want to use the real time timers, you will have to code your NPC AI in the RunAfterInput event (or a particular location if the NPC is static).

Also, because timers are not dependent on player input, the CurrentPlayer object is not valid for IAGEScripts in OnTimer events.

OnAction
The OnAction event runs whenever any player makes a reference to the NPC. The OnAction event takes precedence over all events but Location.OnInput. If a player was to enter "attack gandalf", the OnAction event for Gandalf would fire before the internal engine attack code (all IAGE events take precedence over the inbuilt code).

Before_Display
The before_display event is an inline event for the OnAction event. You must define a procedure with the name "before_display" which will be called by IAGE to look up an alternative for the character's name at runtime.

Eg:

```
proc before_display
   returnvalue = Newname
   cancelevent
end
```

Before_DisplayInitial
This event is identical to the before_display event, except it acts on the characters InitialDescription field instead of the name.

Initialise
The Initialise event is an inline event. You must define a procedure with the name "initialise" which will be called by IAGE when the game is first started to allow you to perform any actions on the object. Note that this event is only called once, and it is called before any players have entered the game.

Each_Turn
This event is fired after every turn that every player makes in the game. These events always run and they are fired straight after the AfterInputImmediate event. This, together with get and set value are very useful for writing daemons.

Example 1. The Thief From Zork
This is relatively simple to do. The thief in Zork moves randomly around the map (AI Mode 2) and when his timer hits, he checks to see if there are any players in his location. If there are, he checks their inventories to see if they have anything which is marked as a treasure (use IAGE "UserBooleans" to set special values like this).

If they are carrying something marked as a treasure, the thief steals it and the user is told the thief has "relieved them of a few valuables".

With the script in it's current form, it only steals the first valuable item from the player. It could easily be adapted to steal every valuable they had.

Here is the script (I have included line numbers so I can refer to specific bits):

```
01 var i = 1
02 var cbi
03
04 while ( i < |currentplayer.count + 1| ) then
05
06     cbi = playerarray(i).getfirstboolitem(treasure)
07     if ( cbi > 0 ) and ( playerarray(i).currentlocation = this.currentlocation ) then
08         playerarray(i).print "The thief pauses to relieve you of some valuables."
09         item(cbi).currentlocation = this.containerlocation
10     endif
11     i = |i + 1|
12 endwhile
```

First of all, line 1 creates a variable called “i” and assigned a starting value of 1 - this is to be our loopcounter (determining how many times we have looped later on).

Line 2 creates a new variable “cbi” and does not assign it a value.

Line 4 starts a loop which will keep on going until the loopcounter (“i”) reaches the number of players connected to the server plus 1. Why not until the number of players connected? If it did, the last player would not be looked at because the while condition would be looked at, see that it was on the last player and not execute the loop code.

Line 6 Calls a special player method called “getfirstboolitem” - this returns the ID of the first item the player is carrying which has that UserBoolean value.

Line 7 then checks to see if the item returned from GetFirstBoolItem was 0. A zero will be returned if the player has no items with that boolean. The other part of the condition makes sure that this player is in the same location as the NPC. It wouldn’t be very fair to steal things from players when they aren’t even in the same location would it?

Lines 8 and 9 obviously only run if line 7 is true and they output a message to the player, telling them they got robbed and moves the item they had stolen to the thief’s pockets.

Finally, line 11 is the loop iterator - we have to have this in, otherwise IAGE would lock into an endless loop.

Player Object (Multi-Instance)
PlayerArray Object (Multi-Instance)
CurrentPlayer Object (Single-Instance)

The Player object collection contains information about all the connected players in a game.

There are actually three different methods of accessing a player object, however all the properties and methods remain the same. These are:

CurrentPlayer.<property or method>

This returns the player which initiated this code run. So, if player 1 sends a “get all” command or something, they fire off the location afterinput code, the runafterinput code and finally the internal code. Because they fired it off, they are the current player (hence the object name!). It is important to note that there is one special event in IAGE where the currentplayer is not valid and that is the NPC OnTimer event, since it runs independently, no player is ever reponsible for firing the timer code.

Player(X).<property or method>

References a player by it’s Index property. Useful for referencing player objects from NPC “AttackingWho” and various other IAGE properties since Index is always used to link them together.

PlayerArray(X).<property or method>

Because the index is an incrementing counter from when the server started, indexes may not start at one and will go higher than the number of players connected. If you want to iterate through the players on the system, you need something that does start at one and finish at the number of connected players. This is what playerarray gives you - an array of players, indexed by an arbitrary number between 1 and player.count.

Properties	Methods	Events
Count	ShowScore	(none)
Index	ShowStatus	
Name	Print	
DisplayName	GetFirstBoolItem	
Score	DropAllItems	
Turns	IsCarrying	
CurrentLocation	DisplayCurrentLocation	
IPAddress	GetValue	
CanSee	SetValue	
OutputToPlayer	Quit	
State	SaveState	
StateItem	RestoreState	
LastNoun	AskParserQuestion	
LastCommand		
WeightCarried		
ItemsCarried		
SizeCarried		
VerboseMode		
HitPoints		
DamageIndicator		
ChanceOfHitting		
Money		
TextOnly		
CP1-CP30		

Count
Returns the number of players currently connected to the game server.

Index
The unique index of this player.

Name
The alias this player is using.

DisplayName
The alias output to other players in location descriptions.

Score
The player's current score.

Turns
The player's number of turns taken.

CurrentLocation
The ID of the location this player is currently in.

IPAddress - The IP Address of this player.

CanSee
Returns true or false, depending on whether the player can see (if they are in a dark location and have a lightsource available).

OutputToPlayer
Returns true or false, depending on whether any text has been transmitted to the player since their last input.

State
Numeric property containing one of 4 settings:

1 - Normal
2 - Sitting
3 - Lying
4 - Stood On

StateItem
Used with State, the item the player is currently affecting. If they are stood on a bed for example and that is item 5, StateItem would return 5 and State would be 4.

LastNoun
ID of the last noun the player used. Used internally by the parser for “it”

LastCommand
A string containing the last sentence the player transmitted. Used internally by the parser for “again”

WeightCarried
The total weight this player is carrying.

ItemsCarried
The total number of items this player is carrying.

SizeCarried
The total size this player is carrying.

VerboseMode
Determines whether the player is in verbose mode (true or false). If true, location descriptions will be repeated for locations already visited.

HitPoints
Number of hit points the player has remaining.

DamageIndicator
The maximum number of hit points a player can take from an NPC/other player without a weapon.

ChanceOfHitting
The percentage chance this player has of hitting their target in combat.

Money
The amount of money this player is carrying (only if using IAGE money).

CP1-CP30
Custom properties. The are variant properties (can hold numbers, strings or booleans) relating directly to this player. For example, you could nominate custom property 1 (CP1) as a strength indicator and CP2 as dexterity for a more role-playing sort of feel.

Quit
Makes the player quit, just as if they typed it at the console.

ShowScore
Displays the player's score to them (by firing the OnScore event of the game object).

ShowStatus
Displays the player's combat status to them (as if they typed "status" at the console).

Print
Displays the expression to the player. Mathematics and concatenation, or indeed any valid expression is allowed. For more info on expressions, look up expressions in this script reference.

Eg: playerarray(2).print This^is^pretty^cool^isn't^it?
Your^server^index^is^_&_playerarray(2).index

GetFirstBoolItem
Accepts a UserBoolean name, then looks through the player's inventory for the first item which has that UserBoolean. If it finds one, the item's ID is returned. If none are found, zero is returned.

Eg:

```
var iid = currentplayer.getfirstboolitem(isgun)
```

DropAllItems
Forces the player to drop all their items.

IsCarrying
Accepts an expression which must return an Item ID. Returns true or false according to whether the player is carrying that item.

Eg:

```
if ( currentplayer.iscarrying(10) ) then
```

DisplayCurrentLocation
Displays this player's current location to them.

SetValue
Accepts a bracketed argument representing a name and allows you to assign a value to it, effictively allowing you to attach persistable variables to players aside from the CP properties. If a value with the name you give exists, it is updated.

Eg:

currentplayer.setvalue(hasbignose) = You^sure^do!

GetValue
Allows you to retrieve a named value from a player object.

Eg:

currentplayer.print currentplayer.getvalue(hasbignose)

TextOnly
Allows you to set, or get whether this player is in text only mode (a mode initiated by clients which cannot handle images or the other IAGE media). Both the ServPlet and client applet use text only mode.

SaveState
Accepts two arguments, one representing a unique identifier and the other for a password. This routine then saves the current player's position (in multiplayer games), or a snapshot of the whole game world (single player games).

Eg:
currentplayer.savestate 00 trousers

Saves the current player's game with the identifier 00 and the password trousers. To restore the game again, the player will have to make sure they are using the same alias and do restorestate 00 trousers. (See the IAGE library code in the StandardLib codemodule to see how this works).

RestoreState
Same syntax as SaveState, but used to reload a saved position. Again, see the IAGE library code. Note that RestoreState sets ReturnValue to be true if the restore was a success.

AskParserQuestion
Tells the parser that the last text printed to this player asked a question that requires them to respond with something. This is commonly used by the library, but you can make use of it yourself. It requires an argument that states which noun is required - 1 to 3, or 4 if you want to ask for an adverb.
Eg. Say the player is going to throw a stone, you put some code in the stone to ask for a target (noun2 - "THROW STONE AT X") if the player does not supply one:

```
;throw
        if ( input.noun2 = 0 ) then
                currentplayer.print |"What do you want to throw the " & this.thename & " at?"|
                askparserquestion 2
```

Flag Object (Multi-Instance)

The flag object collection contains information about all the flags in a game.

Properties

Count
Value

Count
Returns the number of flags in the game.

Value
The value of this flag. Flag values are variants - they can hold strings, numbers, arrays and boolean (true/false) types.

Game Object (Single-Instance)

The game object contains useful game methods and events.

Methods
DisplayCurrentLocation
DisplayVersion
GetInventory

Events
OnScore
OnQuit
OnDisplayBanner
OnStart
RunBeforeInput
RunAfterInput

DisplayCurrentLocation
Displays the current player's location to them.

DisplayVersion
Displays the game version to the current player.

GetInventory
Accepts an expression evaluating to a location ID and returns a string containing a list of all the objects in that location ID (including players, NPCs, containers and surface containers).

OnScore
Runs when a score is requested by a player. Relies on the IAGE script in it to return a string displaying the score as the returnvalue. The template has a default script for generating a score.

OnQuit
Runs when a player quits.

OnDisplayBanner
Runs whenever a banner request is made by a player (the default is score / turns in the template). The banner is the top right text of the client.

OnStart
Runs when a player enters the game.

RunBeforeInput
Runs just before control is return to a player ready for further input.

RunAfterInput
Runs after a player has transmitted their input and after all other events have been run.

Input Object (Single-Instance)

The input object contains information about a player's most recent input.

Properties	**Methods**
Noun	Totalwords
Noun2	word
Noun3	contains
SNoun	
SNoun2	
SNoun3	
Verb	
SVerb	
Verb2	
SVerb2	
Verb3	
SVerb3	
Adverb	
SAdverb	
Adverb2	
SAdverb2	
Adverb3	
SAdverb3	
IsNPCAddress	

<dict>, <SDict>

Noun, Noun2, Noun3, Verb, Adverb etc. all return the ID of the dictionary item they represent as entered by the player. So if the player entered GET ALL, then verb would be 5 (for get) and Noun would be 2 (for all).
The values prefixed with S return the actual text of those words.

IsNPCAddress
Returns true if the input was addressing an NPC in the form <npcname>, <command>

Totalwords
Returns the number of words entered by the player.

Word(x)
Where x is a word number from 1 to Totalwords - returns the text of that word entered by the player.

Contains(expr)
Where expr evaluates to a string, checks to see if this string appears anywhere in the players input and returns true if it does.

Internal Object (Single-Instance)

The internal object allows you to directly call IAGE internal routines.
All routines follow the same syntax - Internal.[method] [arg1expr] [arg2expr] etc. and all routines act as if the currentplayer performed the action.

Methods

AttackPlayer	LieOn
AttackNPC	Light
Close	Open
Drop	Put
Eat	Read
Examine	Remove
Extinguish	SitOn
Get	Stand
GetIn	Unwear
GetOutOf	Wear
Go	
Inventory	

AttackPlayer <PlayerIndex> [with <Item ID>]
AttackNPC <NPC ID> [with <Item ID>]
Close <Item ID>
Drop <Item ID>
Eat <Item ID>
Examine <Item ID>
Extinguish <Item ID>
Get <Item ID> or <All>
GetIn <Item ID>
GetOutOf <Item ID>
Go <Direction - n,s,e,w,u,d,ne,se,nw,sw>
Inventory
LieOn <Item ID>
Light <Item ID>
Open <Item ID>
Put <Item ID> [in <Item ID>]
Read <Item ID>
Remove <Item ID> [from <Item ID>]
SitOn <Item ID>
Stand
Unwear <Item ID>
Wear <Item ID>

Item Object (Multi-Instance)

The item object collection contains information about all the items in a game.

Properties	Methods	Events
Count	GetUserBoolean	OnAction
ID	AddUserBoolean	
Name	RemoveUserBoolean	Inline (OnAction)
TheName	GetValue	
CurrentLocation	SetValue	Initialise
Description	GetValueByExp	After_Get
DefaultExamine	SetValueByExp	After_Drop
Weight	CodeClear	After_Insert
ContainedWeight	CodeAddLine	After_Remove
Size	FireInLineEvent	Before_Display
ContainedSize		Before_DisplayInitial
IsLightSource		Before_Get
IsLit		Before_Drop
IsWearable		Before_Insert
IsWorn		Before_Remove
Invisible		Each_Turn
IsContainer		
IsWeapon		
IsFixed		
FixedMessage		
HasSurface		
CanBeStoodOn		
CanBeSatOn		
CanBeLaidOn		
CanOpenClose		
OpenCloseState		
IsEdible		
EdibleHitPoints		
IsReadable		
ReadableText		
IsSubItem		
SubItemOf		
DamageIndicator		
MovedFromOriginalLocation		

Count
Returns the number of items in the game.

ID
The unique identifier of this item.

Name
The name of this item. This will output to the player in inventories and containers. This will also be output in location descriptions if MovedFromOriginalLocation is true.

TheName
The name of this item without the singular/plural prefix. Use this when you wish to refer to the item with "the" - eg. Name = a brass lamp, TheName = brass lamp

CurrentLocation
The current position of this item.

Description
The initial description of the item, prior to it being moved. If this is blank, the name will be used.

DefaultExamine
The default examine message given when a player examines the item. If blank, a "You notice nothing special about the x" will be generated.

Weight
Whole number representing the weight of the item.

ContainedWeight
The total weight of items inside this item if it is a container (including containers within those containers).

Size
Whole number representing the size of the item.

ContainedSize
The total size of items inside this item if it is a container.

IsLightSource
True if this object can be lit or extinguished.

IsLit
True if this object is providing light.

IsWearable
True if this item can be worn.

IsWorn
True if this item is worn by the player carrying it.

Invisible
True if this item is invisible. Invisible items are not displayed in location descriptions, however they can be manipulated like normal items.

IsContainer
True if this item can contain other items.

IsWeapon
True if this item can be used as a weapon.

IsFixed
True if this item cannot be moved from it's starting location.

FixedMessage
The message to give if a player attempts to move this item from it's starting location and it's fixed.

HasSurface
True if this item can hold items on top of it (eg: a table).

CanBeStoodOn
True if this item can be stood on by players.

CanBeSatOn
True if this item can be sat on by players.

CanBeLaidOn
True if this item can be laid on by players.

CanOpenClose
True if this item can be open or closed.

OpenCloseState
True if this item is open.

IsEdible
True if this item can be eaten.

EdibleHitPoints
The number of hit points awarded to a player for eating this item.

IsReadable
True if this item can be read.

ReadableText
The text to show when the item is read.

IsSubItem
True if this item belongs to another item (eg. is a component of. For example, a TV set has an on/off switch. The switch would be a sub item of the TV).

SubItemOf
The ID of the item this is a subitem of

DamageIndicator
The maximum amount of hit points this item can take if used as a weapon.

MovedFromOriginalLocation
True if this item has been moved by a player or NPC since the game started.

GetUserBoolean
Accepts a boolean name and returns true if this item has that user boolean attached.

Eg:

```
if ( leaflet.getuserboolean(treasure) = true ) then
```

AddUserBoolean
Accepts a boolean name and adds it to the specified object.

Eg:

```
jeweledegg.adduserboolean(treasure)
```

RemoveUserBoolean
Accepts a boolean name and removes it from the specified object.

Eg:

```
jeweledegg.removeuserboolean(treasure)
```

SetValue
Accepts a parameter name and allows you to assign a named value to this item. If a value exists with that name, it is updated, otherwise it is created.

Eg:

```
leaflet.setvalue(doorcount) = 58
```

GetValue
Accepts a parameter name and returns the value for this named argument.

Eg:

```
currentplayer.print leaflet.getvalue(doorcount)

if ( leaflet.getvalue(doorcount) = 58 ) then
  currentplayer.print “It’s 58!”
endif
```

OnAction
The OnAction event fires when this item is referenced by a player (ie. the item is named in player input). If a player's input contains references to multiple items, they will be executed in turn according to the order the player entered them.

Initialise
The Initialise event is an inline event. You must define a procedure with the name "initialise" which will be called by IAGE when the game is first started to allow you to perform any actions on the object. Note that this event is only called once, and it is called before any players have entered the game.

Before_Display
The before_display event is an inline event for the OnAction event. You must define a procedure with the name "before_display" which will be called by IAGE to look up an alternative for the item's name at runtime.

Eg:

```
proc before_display
   returnvalue = Newname
   cancelevent
end
```

Before_DisplayInitial
This event is identical to the before_display event, except it acts on the item's InitialDescription field instead of the name.

Before_Get
This syntax is identical to the before_display event. This event occurs when the IAGE take routine is fired to pick up an item. The returnvalue is the message to show instead of the usual "Taken." message and if you do not want the user to actually take the object, call CancelEvent.

Before_Drop
The syntax for this event is the same as the before_display event. This event occurs when the IAGE drop routine is fired to drop an item. The returnvalue is the message to show instead of the usual "Dropped." message and if you do not want the user to actually drop the object, call CancelEvent.

Before_Insert
The syntax for this event is the same as the before_display event. This event occurs when the IAGE put routine is fired to put this item in or on a container. The returnvalue is the message to show instead of the usual "Done." message and if you do not want the user to actually insert the object, call CancelEvent.

Before_Remove
The syntax for this event is the same as the before_display event. This event occurs when the IAGE remove routine is fired to remove this item from a container. The returnvalue is the message to show instead of the usual "Done." message and if you do not want the user to actually remove the object, call CancelEvent.

After_Get, After_Drop, After_Insert, After_Remove
These inline events are fired after an item is taken by the player, dropped, put in or on a container, or removed from a container respectively. Because they occur afterwards, the normal messages and actions are carried out first.

Each_Turn
This event is fired after every turn that every player makes in the game. These events always run and they are fired straight after the AfterInputImmediate event. This, together with get and set value are very useful for writing daemons.

Location Object (Multi-Instance)

The location object collection contains information about all the locations in a game.

Properties	Methods	Events
Count	GetValue	OnInput
ID	SetValue	OnDisplay
Name	CodeClear	
ImagePath	FireInlineEvent	Inline(OnDisplay)
Description	CodeAddLine	
IsDark		Before_Display
Compass Directions		
		Inline(OnInput)
		Each_Turn
		Initialise

Count
Returns the number of locations in the game.

ID
The unique identifier of this location.

Name
The location name. This is also output before the description.

ImagePath
The absolute path to an image associated with this location. You may specify a relative path if you set the prefix in the Game.Mediabase property.

Description
The text of the location description.

IsDark
True if the current location has no lightsource of it’s own.

Compass Directions (N, S, E, W, U, D, NE, NW, SE, SW)
Gets/sets a location ID of the location the player will get to by entering that direction.

SetValue
Accepts a parameter name and allows you to assign a named value to this item. If a value exists with that name, it is updated, otherwise it is created.

Eg:

```
path.setvalue(doorcount) = 58
```

GetValue
Accepts a parameter name and returns the value for this named argument.

Eg:

```
currentplayer.print path.getvalue(doorcount)

if ( path.getvalue(doorcount) = 58 ) then
   currentplayer.print “It’s 58!”
endif
```

OnInput
The OnInput event runs after a player enters something in this location.

OnDisplay
The OnDisplay event runs when this location’s description is displayed to a player and before any objects, npcs and other players are listed. This event is useful for “one off” remarks in locations.

Before_Display
The before_display event is an inline event for the OnDisplay event. You must define a procedure with the name “before_display” which will be called by IAGE to look up an alternative for the item’s name at runtime.

Eg:

```
proc before_display
   returnvalue = Newname
   cancelevent
end
```

Initialise
The Initialise event is an inline event. You must define a procedure with the name “initialise” which will be called by IAGE when the game is first started to allow you to perform any actions on the object. Note that this event is only called once, and it is called before any players have entered the game.

Each_Turn
This event is fired after every turn that every player makes in the game. These events always run and they are fired straight after the AfterInputImmediate event. This, together with get and set value are very useful for writing daemons.

Message Object (Multi-Instance)

The message object collection contains information about all the messages in a game.

Properties	Methods
Count	Show
ID	ShowOthers
Text	ShowAllIn

Count
Returns the total number of messages in the game.

ID
The unique identifier of this message.

Text
The text of this message.

Show
Shows this message to the current player.

ShowOthers
Shows this message to everyone but the current player.

ShowAllIn
Accepts an expression which evaluates to a location ID. Shows this message to everyone in that location except the current player.

Eg:

message(53).showallin westofhouse.id