**PunyInform**

*An Inform library for writing small and fast text adventures.*

Version 1.0 (beta 1), 21 May 2020

Written by: Johan Berntsson, Fredrik Ramsberg, Pablo Martínez and Tomas Öberg

# TABLE OF CONTENTS

* [Introduction](#Introduction)
* [Comparison with the Inform Standard Library](#X59fc6137816c18d49fe76c95228e29f0fa4c5d5)
  + [Getting Started](#Getting-Started)
  + [Articles](#Articles)
  + [Daemons and timers](#Daemons-and-timers)
  + [Library Messages and Customization](#Library-Messages-and-Customization)
  + [Direction Handling](#Direction-Handling)
  + [Darkness](#Darkness)
  + [Scoring](#Scoring)
  + [Box Statements and Menus](#Box-Statements-and-Menus)
  + [Parser](#Parser)
* [Programming Advice](#Programming-Advice)
  + [Customizing the library](#Customizing-the-library)
  + [Limitations for z3](#Limitations-for-z3)
  + [Properties](#Properties)
* [List of Routines](#List-of-Routines)
  + [Library Routines](#Library-Routines)
  + [Library Entry Routines](#Library-Entry-Routines)
  + [Additional Public Routines](#Additional-Public-Routines)
  + [PunyInform Public Routines](#PunyInform-Public-Routines)
* [List of Properties](#List-of-Properties)
* [List of Variables](#List-of-Variables)
* [List of Attributes](#List-of-Attributes)
* [List of Constants](#List-of-Constants)
* [Grammar](#Grammar)
* [Extensions](#Extensions)
  + [flags](#flags)
  + [cheap\_scenery](#cheap_scenery)

# Introduction

PunyInform is a library written in Inform which allows people to create text adventures/interactive fiction using the Z-machine virtual machine.

The main goal of PunyInform is to allow for games which are fast and have a small memory footprint. This should make the games run well on older architectures, such as the 8 bit computers of the 1980s. Our main target is to make it suitable for games on the Commodore 64 using Ozmoo (https://github.com/johanberntsson/ozmoo)

PunyInform is based on the Inform 6 standard library, developed by Graham Nelson. In this document DM4 refers to the *Inform* *Designer’s Manual, version 4*, which is availble online at: <http://www.inform-fiction.org/manual/html/index.html>

A PunyInform game can be compiled to z3, z5 or z8, but lacks support for Glulx. To compile games using PunyInform, we recommend the official Inform 6 compiler maintained by David Kinder, at <https://github.com/DavidKinder/Inform6> . We are using version 6.33 for all development work. Newer versions are highly likely to work fine as well.

We want to thank these people for supporting the development of PunyInform

* Graham Nelson: for giving his blessing to our project and the use of the PunyInform name
* David Kinder and Andrew Plotkin: for improving z3 support in Inform
* Pablo Martínez: for patches and making the first full-size game in PunyInform
* Tomas Öberg: for patches and encouragement

# Comparison with the Inform Standard Library

A game written in PunyInform is very similar to a game written with the Inform standard library with the exception of which files to include and where to place code that customize the library. However, there are some major changes that are documented in this section.

## Getting Started

You can use the minimal.inf file, supplied with PunyInform, as a starting point for developing a new game.

The general pattern of a PunyInform game is:

! define library globals here  
  
 Include "globals.h";  
  
 ! add extension routines and other library customizations here  
  
 Include "puny.h";  
  
 ! add normal game code here  
  
 [Initialise;  
 ];

The library variables Story, Headline, MAX\_SCORE, NUMBER\_TASKS, TASKS\_PROVIDED, AMUSING\_PROVIDED, MAX\_CARRIED, and SACK\_OBJECT should be defined before including globals.h, if needed. These variables are documented in DM4.

Library customization, such as supplying a library extension point like PrintTask, goes between the globals.h and puny.h inclusions.

After the includes you add game code and an Initialise routine, as in other Inform games.

## Articles

PunyInform, unlike the Inform standard library, will not figure out when it should be article “an”. You need to specify it using the article property every time it should be “an”. Example:

Object Umbrella "umbrella"  
 with  
 name 'umbrella',  
 article "an";

Another difference is that PunyInform doesn’t support the articles (note the s) property. This was only added to the Inform library because it’s useful for some languages other than English.

## Daemons and timers

Property daemon is an alias for property time\_out. This means you can’t have a daemon and a timer on the same object. If you want both, put one of them in another object, possibly a dummy object whose only purpose is to hold the timer/daemon.

## Library Messages and Customization

All system messages that can be replaced can be found in the file messages.h.

PunyInform uses two form of library messages: static strings and complex messages. A typical static string is “Taken.”. If a message has parts that vary, if the same message should be shared by several different message identifiers, or a newline should NOT be printed after the message, the message needs to be a complex message. A complex message has its own piece of code to print it.

Each message is defined as either a static string or a complex message in messages.h. If you want to replace a message, you can choose to replace it with a static string or a complex message, regardless of its type in messages.h. You do this by defining constants and possible a LibraryMessages routine before the inclusion of puny.h.

To replace a message with a static string, define a constant with the same name as the message identifier and give it a string value, i.e:

Constant MSG\_INSERT\_NO\_ROOM "It's kinda full already, I'm afraid.";

To replace a message with a complex message, define a constant with the same name as the message identifier, give it a value >= 1000 and provide a LibraryMessages routine to handle it, i.e:

Constant MSG\_EXAMINE\_NOTHING\_SPECIAL 1001;  
  
[LibraryMessages p\_msg p\_arg\_1;  
 switch(p\_msg) {  
 MSG\_EXAMINE\_NOTHING\_SPECIAL:  
 print (The) noun, " looks perfectly normal in every way.";  
 rtrue;  
 }  
 rfalse;  
];

The LibraryMessages routine takes two arguments - a message identifier (p\_msg) and an optional argument which a few messages use (p\_arg\_1). Make sure the routine returns true after printing a message, and false if it didn’t print anything.

## Direction Handling

The Compass and the twelve direction objects, as described in DM4, are not available in PunyInform. Instead, there is a single object called Directions and two global variables called selected\_direction and selected\_direction\_index. When compiling games for the z3 format, a game can have a maximum of 255 objects. With this in mind, it’s good to use a single object for directions instead of 13 objects.

Whenever the player has typed a direction, noun is Directions and selected\_direction contains the property number for the direction the player typed. If the player didn’t type a direction, these variables will be 0. The name of the Direction object is always the currently selected direction, or “unknown direction” if no direction is selected. So, to implement a robot which will stop the player from going north or east, one might write a react\_before routine like this:

Object Robot "Floyd"  
 with  
 react\_before [;  
 Go:  
 if(selected\_direction == n\_to or e\_to)   
 "~My mother always told me to avoid going ", (name) Directions, ".~, says Floyd.";  
 ],  
 has animate;

selected\_direction\_index is something you will probably use less often, but it can nevertheless be useful in some cases. You can use it to look up the dictionary words which can be used to refer to that direction, the property number and the name of the direction:

print (address) abbr\_direction\_array-->selected\_direction\_index; ! prints the short dictionary word, like 'n//'  
print (address) full\_direction\_array-->selected\_direction\_index; ! prints the long dictionary word, like 'north'  
print direction\_properties\_array-->selected\_direction\_index; ! prints the property number, like 7  
print (string) direction\_name\_array-->selected\_direction\_index; ! prints the direction name, like "north"

Each of these arrays is a table, so all of them have the number of directions as element 0. This is useful if you’re writing a library extension and want to iterate over all directions in a safe manner.

### Fake direction objects.

For each direction, there is also a fake direction object: FAKE\_N\_OBJ, FAKE\_SW\_OBJ, FAKE\_OUT\_OBJ etc. If you need to generate an action in code which has a direction in it, this requires using the corresponding fake direction object, like this:

<<Go FAKE\_N\_OBJ>>;  
<<Push Stone FAKE\_NW\_OBJ>>;

If you want to go in a direction and you know the property number for that direction, you can find the corresponding fake direction object by subtracting n\_to from the value and adding FAKE\_N\_OBJ, like this:

dir\_prop = ne\_to; ! Or any direction you like  
fake\_obj = dir\_prop - n\_to + FAKE\_N\_OBJ; ! Note: n\_to and FAKE\_N\_OBJ are part of the forumla. Don't change!   
<<Go fake\_obj>>;

Each fake direction object is just a constant. PunyInform recognizes these constants and sets selected\_direction and selected\_direction\_index properly. This is, as far as we can tell, the only use for the fake direction objects.

## Darkness

PunyInform uses a simplified concept of darkness. Instead of putting the player in a special TheDark object when in darkness and keeping real\_location updated, as described in DM4, PunyInform keeps a global variable “darkness” and updates the scope accordingly.

A game using PunyInform should check “darkness” to see if there is light.

## Scoring

Scoring works as in DM4 with the exception of the scored attribute, which isn’t supported. Because of this the OBJECT\_SCORE library variable isn’t used.

## Box Statements and Menus

The box statement is not available in version 3 games, and the usual menu extensions will not work either since version 3 games lack cursor control commands. Instead PunyInform provides extensions that approximate this functionality. See the Extensions section for more detail and how to enable these routines.

## Parser

The parser is to a large extent compatible with Inform, for example wn, NextWord() and NextWordStopped() are implemented, and noun/second/inp1/inp2/special\_number/parsed\_number work the same.

General parse routines are supported with the exception of GRP\_REPARSE which isn’t supported. The reason for this is that version 3 games cannot retokenise the input from the reconstructed string.

# Programming Advice

The Inform standard veneer routine for printing informative messages for all sorts of runtime errors that can occur is replaced with a simpler routine in PunyInform, saving about 1.5 KB. However, the original routine is used when at least one of the constants DEBUG or RUNTIME\_ERRORS is defined.

## Customizing the library

PunyInform is designed to be as small as possible to run well on old computers, and some features that add to the size have made optional. If you want to enable these features, add a line like “Constant OPTIONAL\_GUESS\_MISSING\_NOUN;” before including globals.h, but keep in mind that it will make the game larger.

The optional parts of PunyLib can be enabled with these constants:

|  |  |  |
| --- | --- | --- |
| **Option** | **Size** | **Comment** |
| OPTIONAL\_ALLOW\_WRITTEN\_NUMBERS |  | to be able to parse “one”, “two” etc as numbers. |
| OPTIONAL\_DEBUG\_VERBS |  | enable some debugging verbs for game development. These include ‘scope’, ‘random’ and ‘pronouns’. |
| OPTIONAL\_EXTENDED\_VERBSET |  | add a set of less important, but nice to have, verbs in the grammar. |
| OPTIONAL\_GUESS\_MISSING\_NOUN | 512 bytes | add code to guess missing parts of an incomplete input, such as a door when typing only ‘open’, and accepting the input with a “(assuming the wooden door)” message. |

Please note that if you compile your game in DEBUG mode with the -D switch to the Inform compiler, then OPTIONAL\_DEBUG\_VERBS are automatically enabled. But to have access to debug verbs in release mode you need to define OPTIONAL\_DEBUG\_VERBS manually in your game.

PunyInform can also use a set of standard abbreviations to make strings more compact. If you want to provide your own abbreviations, define the constant CUSTOM\_ABBREVIATIONS in your game. Keep in mind that you need to compile with the “-e” flag to make the compiler use abbreviations.

## Limitations for z3

If you want to compile a game to z3 format, this is what you need to keep in mind:

* A game can use no more than 32 attributes and 30 common properties. PunyInform defines 28 attributes and 29 common properties.
* Arrays in common properties can only hold four values. Arrays in individual properties however, can hold 32 values.
* When using message passing (like “MyBox.AddWeight(5)” ), no more than one argument may be passed. (In regular Inform, message passing doesn’t work at all in z3.)
* Dynamic object creation and deletion can not be used.
* If you need more than four names for an object in a z3 game, give it a parse\_name routine.

When the player is inside an object, in a z5 game, the library will print the name of the object on the statusline, in definite form (“The box”). In a z3 game, the object name string will be printed as-is, typicall like “box”. This behavior in z3 games is part of the Z-machine specification. If you want a z3 game to print a different name for when the player is inside the object, you can set the object name string to the desired name, and override it with short\_name for all other uses, like this:

Object box "The box"  
 with short\_name "box"  
 has container openable enterable;

## Properties

A property can be used to store a 16-bit value, or an array of values. In z5, a property array can hold up to 32 values. In z3, a property array can only hold 4 values if it is in a common property but 32 values if it is in an individual property.

If a property is declared as additive, the values for an object are concatenated with the values of its class, if any, and put into an array.

A property can either be common or individual. Common properties are a little faster to access and use a little less memory than individual properties. A z5 or z8 game can use a maximum of 62 common properties, while a z3 game can use a maximum of 30 common properties. PunyInform uses 29 common properties, so if you’re building a z3 game, you can only add a single common property. The value of a common property can always be read, but it can only be written if it has been included in the object declaration. If you don’t include it, there is no memory allocated to store a value. If you read the value of such a property, you just get the default value (typically 0).

A common property is created by declaring it with

Property *propertyname*;

To access a property, you write object.\_propertyname\_, like this:

Dog.description = “The dog looks sleepy.”;

To check if an object has a value for a property (to see if it can be written if it is a common property or to see if it can be read or written if it is an individual property, use *provides*:

If(Dog provides description) …

# List of Routines

PunyInform defines both public and private routines. The private routines are prefixed with an underscore (for example, \_ParsePattern) and should not be used by a game developer. The public routines do not have this prefix, and are for general use. Most of the public routines are same, or very similar, to corresponding routines in DM4, but PunyInform also offers a few extra routines not available in Inform. All public routines are listed below in this section.

## Library Routines

These library routines are supported by PunyInform, as described in DM4.

|  |  |
| --- | --- |
| **Library Routine** | **Comment** |
| CommonAncestor |  |
| DrawStatusLine | Not available in version 3 games |
| IndirectlyContains |  |
| InScope |  |
| LoopOverScope |  |
| NextWord |  |
| NextWordStopped |  |
| NumberWord |  |
| ObjectIsUntouchable |  |
| PlayerTo |  |
| ParseToken |  |
| PlaceInScope |  |
| PronounNotice |  |
| ScopeWithin |  |
| TestScope |  |
| TryNumber |  |
| WordAddress |  |
| WordLength |  |
| YesOrNo |  |

## Library Entry Routines

This library entry routines are supported by PunyInform, as described in the DM4.

|  |  |
| --- | --- |
| **Entry Routine** | **Comment** |
| AfterLife |  |
| AfterPrompt |  |
| Amusing |  |
| BeforeParsing |  |
| DarkToDark |  |
| DeathMessage |  |
| GamePostRoutine |  |
| GamePreRoutine |  |
| InScope | The et\_flag isn’t supported. |
| LookRoutine |  |
| NewRoom |  |
| ParseNumber |  |
| PrintTaskName |  |
| PrintVerb |  |
| TimePasses |  |
| UnknownVerb |  |

These library entry routines are not supported

|  |  |
| --- | --- |
| **Entry Routine** | **Comment** |
| ChooseObjects | The parser internals differ too much |
| ParserError | The parser internals differ too much |

## Additional Public Routines

|  |  |
| --- | --- |
| **Routine Name** | **Comment** |
| PrintOrRun |  |
| RunRoutines |  |
| CTheyreorThats |  |
| ItorThem | Print directive |
| IsOrAre | Print directive |

## PunyInform Public Routines

|  |  |
| --- | --- |
| **Routine Name** | **Comment** |
| OnOff | Print directive |
| ObjectIsInvisible | Similar to ObjectIsUntouchalbe (DM4) |
| PrintMsg |  |
| RunTimeError |  |

# List of Properties

These are the properties defined by the library:

|  |  |
| --- | --- |
| **Property** | **Comment** |
| add\_to\_scope |  |
| after |  |
| article |  |
| before |  |
| cant\_go |  |
| capacity |  |
| d\_to |  |
| daemon |  |
| describe |  |
| description |  |
| door\_dir |  |
| e\_to |  |
| found\_in |  |
| grammar |  |
| in\_to |  |
| initial |  |
| inside\_description |  |
| invent |  |
| life |  |
| list\_together |  |
| n\_to |  |
| name |  |
| ne\_to |  |
| number |  |
| nw\_to |  |
| orders |  |
| out\_to |  |
| parse\_name |  |
| plural |  |
| react\_after |  |
| s\_to |  |
| se\_to |  |
| short\_name\_indef |  |
| short\_name |  |
| sw\_to |  |
| time\_left |  |
| u\_to |  |
| w\_to |  |
| when\_closed |  |
| when\_open |  |
| with\_key |  |

# List of Variables

These variables are the same as in DM4.

|  |  |
| --- | --- |
| **Variable** | **Comment** |
| action |  |
| actor |  |
| articles |  |
| consult\_from |  |
| consult\_words |  |
| deadflag |  |
| herobj |  |
| himobj |  |
| inp1 |  |
| inp2 |  |
| itobj |  |
| keep\_silent |  |
| location |  |
| lookmode |  |
| parsed\_number |  |
| parser\_action |  |
| scope\_stage |  |
| score |  |
| second |  |
| special\_number |  |
| verb\_word |  |
| verb\_wordnum |  |
| wn |  |

These variables are PunyInform only.

|  |  |
| --- | --- |
| **Variable** | **Comment** |
| darkness |  |

These variables are used in the Inform standard library and are listed in DM4, but are not used in PunyInform.

|  |  |
| --- | --- |
| **Variable** | **Comment** |
| c\_style |  |
| et\_flag |  |
| inventory\_stage |  |
| listing\_together |  |
| lm\_n |  |
| lm\_o |  |
| notify\_mode |  |
| parser\_one |  |
| parser\_two |  |
| read\_location |  |
| scope\_reason |  |
| standard\_interpreter |  |
| the\_time |  |
| vague\_object |  |

# List of Attributes

These attributes are the same as in DM4.

|  |  |
| --- | --- |
| **Attribute** | **Comment** |
| absent |  |
| animate |  |
| clothing |  |
| concealed |  |
| container |  |
| door |  |
| edible |  |
| enterable |  |
| female |  |
| general |  |
| light |  |
| lockable |  |
| moved |  |
| neuter |  |
| on |  |
| open |  |
| openable |  |
| pluralname |  |
| proper |  |
| scenery |  |
| static |  |
| supporter |  |
| talkable |  |
| transparent |  |
| visited |  |
| workflag |  |
| worn |  |

These attributes are used in the Inform standard library and are listed in DM4, but are not used in PunyInform.

|  |  |
| --- | --- |
| **Attribute** | **Comment** |
| male | not needed, assumed if not female or neuter |
| scored |  |

# List of Constants

These constants are the same as in DM4.

|  |  |
| --- | --- |
| **Constant Name** | **Comment** |
| AMUSING\_PROVIDED |  |
| GPR\_FAIL |  |
| GPR\_MULTIPLE |  |
| GPR\_NUMBER |  |
| GPR\_PREPOSITION |  |
| GPR\_REPARSE |  |
| Headline |  |
| MAX\_CARRIED |  |
| MAX\_SCORE |  |
| MAX\_TIMERS |  |
| NUMBER\_TASKS |  |
| SACK\_OBJECT |  |
| Story |  |
| TASKS\_PROVIDED |  |

These attributes are used in the Inform standard library and are listed in DM4, but are not used in PunyInform. Most of them are parser specific for the standard lib, and the PunyInform parser works differently.

|  |  |
| --- | --- |
| **Constant Name** | **Comment** |
| ANIMA\_PE |  |
| ASKSCOPE\_PE |  |
| CANTSEE\_PE |  |
| DEATH\_MENTION\_UNDO |  |
| EACHTURN\_REASON |  |
| ELEMENTARY\_TT |  |
| EXCEPT\_PE |  |
| ITGONE\_PE |  |
| JUNKAFTER\_PE |  |
| LOOPOVERSCOPE\_REASON |  |
| MMULTI\_PE |  |
| MULTI\_PE |  |
| NO\_PLACES |  |
| NOTHELD\_PE |  |
| NOTHING\_PE |  |
| NUMBER\_PE |  |
| OBJECT\_SCORE |  |
| PARSING\_REASON |  |
| REACT\_AFTER\_REASON |  |
| REACT\_BEFORE\_REASON |  |
| ROOM\_SCORE |  |
| SCENERY\_PE |  |
| SCOPE\_TT |  |
| STUCK\_PE |  |
| TALKING\_REASON |  |
| TESTSCOPE\_REASON |  |
| TOOFEW\_PE |  |
| TOOLIT\_PE |  |
| UPTO\_PE |  |
| USE\_MODULES |  |
| VAGUE\_PE |  |
| VERB\_PE |  |

# Grammar

Here are the standard verbs defined in the library. Verbs that have the “extended” comment are not included by default, but can be added by defining OPTIONAL\_EXTENDED\_VERBSET.

|  |  |
| --- | --- |
| **Verbs** | **Comment** |
| answer say shout speak | - |
| ask | - |
| attack break crack destroy | - |
| blow | extended |
| bother curses darn drat | extended |
| burn light | extended, not in PunyInform |
| buy purchase | extended, not in PunyInform |
| climb scale | - |
| close cover shut | - |
| consult | extended, not in PunyInform |
| cut chop prune slice | - |
| dig | - |
| drink sip swallow | - |
| drop discard throw | - |
| eat | - |
| empty | extended, not in PunyInform |
| enter cross | - |
| examine x | - |
| exit out outside | - |
| fill | - |
| get | - |
| give feed offer pay | - |
| go run walk | - |
| in inside | extended, not in PunyInform |
| insert | - |
| inventory inv i | - |
| jump hop skip | - |
| kiss embrace hug | - |
| leave | - |
| listen hear | - |
| lock | - |
| look l | - |
| no | extended, not in PunyInform |
| open uncover undo unwrap | - |
| peel | extended, not in PunyInform |
| pick | -, not in PunyInform |
| pray | extended, not in PunyInform |
| pry prise prize lever jemmy force | extended, not in PunyInform |
| pull drag | - |
| push clear move press shift | - |
| put | - |
| read | - |
| remove | - |
| rub clean dust polish scrub | - |
| search | - |
| set adjust | extended, not in PunyInform |
| shed disrobe doff | -, not in PunyInform |
| show display present | - |
| shit damn fuck sod | extended |
| sing | extended, not in PunyInform |
| sit lie | -, not in PunyInform |
| sleep nap | extended, not in PunyInform |
| smell sniff | - |
| sorry | extended, not in PunyInform |
| squeeze squash | extended, not in PunyInform |
| stand | - |
| swim dive | -, not in PunyInform |
| swing | extended, not in PunyInform |
| switch | - |
| take carry hold | - |
| taste | - not in PunyInform |
| tell | - |
| think | extended, not in PunyInform |
| tie attach fasten fix | - not in PunyInform |
| touch feel fondle grope | - |
| transfer | extended, not in PunyInform |
| turn rotate screw twist unscrew | - |
| wave | extended, not in PunyInform |
| wear don | - |
| yes y | extended, not in PunyInform |
| unlock | - |
| wait z | - not in PunyInform |
| wake awake awaken | extended |

These are the meta verbs defined in the library.

|  |  |
| --- | --- |
| **Verbs** | **Comment** |
| brief normal |  |
| verbose long |  |
| superbrief short |  |
| notify | not in PunyInform |
| pronouns nouns | OPTIONAL\_DEBUG\_VERBS |
| quit q die |  |
| recording | not in PunyInform |
| replay | not in PunyInform |
| restart |  |
| restore |  |
| save |  |
| score |  |
| fullscore full |  |
| script transcript | not in PunyInform |
| noscript unscript | not in PunyInform |
| verify | not in PunyInform |
| version |  |
| objects | not in PunyInform |
| places | not in PunyInform |

These are the debug verbs defined in the library.

|  |  |
| --- | --- |
| **Verbs** | **Comment** |
| abstract | not in PunyInform |
| actions | not in PunyInform |
| changes | not in PunyInform |
| gonear | not in PunyInform |
| goto | not in PunyInform |
| purloin | not in PunyInform |
| random | OPTIONAL\_DEBUG\_VERBS |
| routines messages | not in PunyInform |
| scope | OPTIONAL\_DEBUG\_VERBS |
| showobj | not in PunyInform |
| showverb | not in PunyInform |
| timers daemons | not in PunyInform |
| trace | not in PunyInform |
| tree | not in PunyInform |

# Extensions

## flags

Flags is a mechanism for keeping track of story progression. If you choose to use flags, four procedures with a total size of about 165 bytes are added to the story file. Also, an eight byte array is added to dynamic memory, plus one byte for every eight flags. All in all this is a very efficient way of keeping track of progress.

If you want to use flags, after including globals.h, set the constant FLAG\_COUNT to the number of flags you need, and then include ext\_flags.h.

You then specify a constant for each flag, like this:

Constant F\_FED\_PARROT 0; ! Has the parrot been fed?  
 Constant F\_TICKET\_OK 1; ! Has Hildegard booked her plane tickets?  
 Constant F\_SAVED\_CAT 2; ! Has the player saved the cat in the tree?

You get the idea – you give each flag a symbolic name so it’s somewhat obvious what it does. Note that the first flag is flag #0, not flag #1.

Setting a flag on or off means calling the routing SetFlag(flag#) or ClearFlag(flag#)

To indicate that the player has saved the cat, call SetFlag(F\_SAVED\_CAT), and to turn off that flag, call ClearFlag(F\_SAVED\_CAT).

Testing a flag is accomplished by calling FlagIsSet or FlagIsClear. So if you have a piece of code that should only be run if the parrot has been fed, you would enclose it in an if(FlagIsSet(F\_FED\_PARROT)) { … }; statement.

Naturally, you can test if a flag is clear by calling FlagIsClear instead.

## cheap\_scenery

This library extension provides a way to implement simple scenery objects which can only be examined, using just a single object for the entire game. This helps keep both the object count and the dynamic memory usage down. For z3 games, which can only hold a total of 255 objects, this is even more important. To use it, include ext\_cheap\_scenery.h after globals.h. Then add a property called cheap\_scenery to the locations where you want to add cheap scenery objects. You can add up to ten cheap scenery objects to one location in this way. For each scenery object, specify, in this order, one adjective, one noun, and one description string or a routine to print one. Instead of an adjective, you may give a synonym to the noun. If no adjective or synonym is needed, use the value 1 in that position.

Note: If you want to use this library extension in a Z-code version 3 game, you must NOT declare cheap\_scenery as a common property, or it will only be able to hold one scenery object instead of ten. For z5 and z8, you can declare it as a common property if you like, or let it be an individual property.

If you want to use the same description for a scenery object in several locations, declare a constant to hold that string, and refer to the constant in each location.

Before including this extension, you can also define a string or routine called SceneryReply. If you do, it will be used whenever the player does something to a scenery object other than examining it. If it is a string, it is printed. If it is a routine it is called. If the routine prints something, it should return true, otherwise false.

If constant DEBUG is defined, the extension will complain about programming mistakes it finds in the cheap\_scenery data in rooms. Without DEBUG, it will keep silent.

Example usage:

[SceneryReply;  
Push:  
 "Now how would you do that?";  
default:  
 rfalse;  
];  
  
Include "ext\_cheap\_scenery.h";  
  
Constant SCN\_WATER = "The water is so beautiful this time of year, all clear and glittering.";  
  
[SCN\_SUN;  
 deadflag = 1;  
 "As you stare right into the sun, you feel a burning sensation in your eyes.  
 After a while, all goes black. With no eyesight, you have little hope of  
 completing your investigations.";  
];  
  
Object RiverBank "River Bank"  
 with  
 description "The river is quite wide here. The sun reflects in the blue water, the birds are  
 flying high up above.",  
 cheap\_scenery  
 'blue' 'water' SCN\_WATER  
 'bird' 'birds' "They seem so careless."  
 1 'sun' SCN\_SUN,  
 has light;