Chapter 7: The standard actions

(Core actions • Actions that reveal information • Actions that print a message • Diverting actions • Communication • Navigation • Miscellaneous actions • Debugging actions)

Core actions

The Dialog standard library contains many pre-implemented actions, but only eighteen of them are capable of modifying the game world. These are the so called *core actions*. The other (non-core) standard actions either reveal information about the world (that you supply via predicates such as (descr $) and (from $ go $ to $)), print stock responses, or divert to one of the core actions.

The core actions are:

[take $Obj]

[drop $Obj]

[wear $Obj]

[remove $Obj]

[put $Obj $Relation $OtherObj]

[open $Obj]

[close $Obj]

[lock $Obj]

[unlock $Obj]

[switch on $Obj]

[switch off $Obj]

[eat $Obj]

[climb $Obj]

[enter $Obj]

[leave $Obj]

[leave $Room $Direction]

[leave $Room $Direction by $Vehicle]

[leave $Room $Direction with $PushedObj]

Common to all the core actions is that their default perform-rules call out to separate narration predicates to print messages to the player. This allows story authors to override only the narration. The narration predicates are queried before the game world is modified. This allows you to print for instance “you pick the apple” instead of “you take the apple” if the apple is still in its original location, e.g. part of the apple tree:

```
(narrate taking
#apple)
```

```
(narrate taking #apple)
 You
 (if) (#apple is pristine) (then) pick (else) take (endif)
 the apple.
```

[Copy to clipboard]

Likewise, you might want to print a particular message when the player is pushing a cart around, without having to dive into the code for updating the position of the cart and player:

```
(narrate leaving $
$Dir with #cart)
```

```
(narrate leaving $ $Dir with #cart)
 The wheels squeak as you push the cart (name $Dir).
```

[Copy to clipboard]

As a rule of thumb, narration predicates are only queried from the eighteen core actions, as part of their default perform-rules. Most other standard actions essentially just print messages; their performance **is** an act of narration. To add flavour to those actions, you must intercept (perform $) directly.

An exception is the action [look $Dir], for looking in a direction. It calls out to (narrate failing to look $Dir) when there's no obvious exit in the given direction.

In order to present a robust and self-consistent world to the player, the story author will want to consider how each of the eighteen core actions might affect the game world. But these considerations only need to be made under certain conditions, as detailed in the following sections.

Items and clothing

If your game has any (item $) or (wearable $) objects, or if the player initially carries or wears anything, consider how those objects are affected by the following core actions:

[take $Obj]

• Makes a query to (narrate taking $Obj).
• Updates the location of $Obj.
• Marks $Obj as handled.

[drop $Obj]

• Makes a query to (narrate dropping $Obj).
• Updates the location of $Obj.
• Marks $Obj as handled.

Items also have a subtle effect on many of the standard actions, such as eating, drinking, and showing things to people: These actions have before-rules for automatically attempting to pick up objects that are required for the action, but only if those objects are items.

If your game has any (wearable $) objects, or if the player initially wears anything, also consider the following core actions:

[wear $Obj]

• Makes a query to (narrate wearing $Obj).
• Updates the location of $Obj.
• Marks $Obj as handled.

[remove $Obj]

• Makes a query to (narrate removing $Obj).
• Updates the location of $Obj.
• Marks $Obj as handled.

Openable or lockable objects

If your game has any (openable $) objects, consider how they are affected by:

[open $Obj]

• Makes a query to (narrate opening $Obj).
• Updates the ($ is closed) flag of $Obj.

[close $Obj]

• Makes a query to (narrate closing $Obj).
• Updates the ($ is closed) flag of $Obj.

If your game has any (lockable $) objects, consider how they are affected by:

[lock $Obj with $Key]

• Makes a query to (narrate locking $Obj with $Key).
• Updates the ($ is locked) flag of $Obj.

[unlock $Obj with $Key]

• Makes a query to (narrate unlocking $Obj with $Key).
• Updates the ($ is locked) flag of $Obj.

Switchable or edible objects

If your game has any (switchable $) objects, consider how those are affected by:

[switch on $Obj]

• Makes a query to (narrate switching on $Obj).
• Updates the ($ is off) flag of $Obj.

[switch off $Obj]

• Makes a query to (narrate switching off $Obj).
• Updates the ($ is off) flag of $Obj.

If your game has any (edible $) objects, consider:

[eat $Obj]

• Makes a query to (narrate eating $Obj).
• Updates the location of $Obj (to nowhere).

Rooms

If your game has any map connections at all, consider:

[leave $Room $Dir]

• Makes a query to (prevent entering $).
• Possibly queries (narrate failing to leave $Room $Dir) and stops.
• Makes a query to (narrate leaving $Room $Dir).
• Updates the location of the player.
• Marks the new room as visited.
• Makes a query to (narrate entering $), which usually diverts to [look].

If your game has any (vehicle $) objects, consider:

[leave $Room $Dir by $Vehicle]

• Makes a query to (prevent entering $).
• Possibly queries (narrate failing to leave $Room $Dir) and stops.
• Makes a query to (narrate leaving $Room $Dir by $Vehicle).
• Updates the location of the vehicle object.
• Marks the new room as visited.
• Makes a query to (narrate entering $), which usually diverts to [look].

If your game has any (pushable $) objects, consider:

[leave $Room $Dir with $Obj]

• Makes a query to (prevent entering $).
• Possibly queries (narrate failing to leave $Room $Dir) and stops.
• Makes a query to (narrate leaving $Room $Dir with $Obj).
• Updates the location of $Obj.
• Marks $Obj as handled.
• Updates the location of the player.
• Marks the new room as visited.
• Makes a query to (narrate entering $), which usually diverts to [look].

Containers and supporters

If your game has any (container $) or (supporter $) objects, consider how they are affected by the following core action:

[put $Obj $Rel $OtherObj]

• Makes a query to (narrate putting $Obj $Rel $OtherObj).
• Updates the location of $Obj.
• Marks $Obj as handled.

If your game has any (actor supporter $) objects, consider:

[climb $Obj]

• Makes a query to (narrate climbing $Obj).
• Updates the location of the player.

If your game has any (actor container $) objects, (door $) objects, or map connections, consider:

[enter $Obj]

• Makes a query to (prevent entering $Obj).
• Makes a query to (narrate entering $Obj).
• Updates the location of the player.

If your game has any (actor supporter $) objects or (actor container $) objects, or if the player is initially on top of an object, or inside a non-room object, consider:

[leave $Obj]

• Makes a query to (narrate leaving $Obj).
• Updates the location of the player.

Actions that reveal information

The following standard actions do not modify the game world, except to reveal hidden objects. Thus, if (#key is hidden) and (#key is #under #rug), then LOOK UNDER RUG will clear the hidden-flag, so that a subsequent GET KEY is allowed to ask: “Did you mean the key under the rug, or the key on top of the table?”

The act of revealing hidden objects isn't listed explicitly in the following table, because it is carried out automatically by the predicates for printing object names, e.g. (a $) and (the $).

[examine $Obj]

• Makes a query to (descr $Obj).
• Makes queries to (appearance $ $ $Obj).

[look]

• Makes a query to (look $).
• Makes queries to (appearance $ $ $).

[exits]

• Displays information from (from $ go $ to $).

[look $Dir]

• Displays information from (from $ go $ to $).
• If there's no exit in that direction, makes a query to (narrate failing to look $Dir).

[look $Rel $Obj]

• Lists children of $Obj having relation $Rel.

[search $Obj]

• Lists children of $Obj (with relations #in, #on, #under, and #behind).

[feel $Obj]

• Makes a query to (feel $Obj).

[inventory]

• Lists any objects held or worn by the current player.

Actions that print a message

The following actions are part of the standard library, but all they do is print stock messages. Those can be error responses or bland statements about how the action had no effect.

[read $Obj]

[listen to $Obj]

[taste $Obj]

[smell $Obj]

[smell]

[kiss $Obj]

[attack $Obj]

[squeeze $Obj]

[fix $Obj]

[clean $Obj]

[cut $Obj with $OtherObj]

[pull $Obj]

[turn $Obj]

[flush $Obj]

[swim in $Obj]

[tie $Obj to $OtherObj]

[talk to $Obj]

[consult $Obj about $Topic]

[greet]

[wait]

[jump]

[dance]

[wave]

[shrug]

[exist]

[sing]

[fly]

[think]

[sleep]

[pray]

[curse]

[wake up]

The following actions require something to be held, so they first attempt to [take $] that object (if it is an item). Then they print a stock message.

[throw $Obj at $OtherObj]

[give $Obj to $OtherObj]

[show $Obj to $OtherObj]

[attack $Obj with $Weapon]

[wave $Obj]

[drink $Obj]

Diverting actions

A number of actions simply divert to other actions, possibly after asking the player for clarification. For instance, [give #money] prints “To whom?”, and sets up an *implicit action*, [give #money to []], with a blank for the missing noun. When the player responds e.g. CLERK, that is understood as the complete action, [give #money to #clerk]. This mechanism will be described in more detail in the chapter on understanding player input.

In some situations, it will be clear from context who is the intended recipient of a [give $] action. Story authors can add rules like the following:

```
(instead of [give $Obj])
 (instead of [give $Obj])
  (current room #store)
  (try [give $Obj to #clerk])
```

[Copy to clipboard]

Most diverting actions are similar to the above example. They ask for a second noun, and set up an implicit action that will receive it:

[give $Obj]

• Diverts to [give $ to $Obj], after asking for clarification.

[show $Obj]

• Diverts to [show $Obj to $], after asking for clarification.

[tie $Obj]

• Diverts to [tie $Obj to $], after asking for clarification.

[cut $Obj]

• Diverts to [cut $Obj with $], after asking for clarification.

[flush]

• Diverts to [flush $], after asking for clarification.

[swim]

• Diverts to [swim in $], after asking for clarification.

[throw $Obj]

• Diverts to [throw $Obj at $], after asking for clarification.

[throw $Obj $Dir]

• Diverts to [throw $Obj at $] or [throw $Obj], after consulting the (from $ go $ to $) predicate.

[hug $Obj]

• Diverts to [kiss $Obj].

[bite $Obj]

• Diverts to [attack $Obj] if $Obj is animate.
• Otherwise diverts to [eat $Obj].

[switch $Obj]

• Diverts to [switch on $Obj] or [switch off $Obj] if $Obj is switchable.
• Otherwise prints a stock message.

[lock $Obj]

• Asks for clarification, unless the correct key is already held.
• Diverts to [lock $ with $].

[unlock $Obj]

• Asks for clarification, unless the correct key is already held.
• Diverts to [unlock $ with $].

The opposite is true for [take $ from $], where the default implementation diverts to the core action [take $] (or [remove $], for worn objects) by removing the extra noun:

[take $Obj from $OtherObj]

• Diverts to [take $Obj] or [remove $Obj] if $Obj is a child of $OtherObj.
• Otherwise prints a stock message.

LISTEN (without a noun) diverts to an action for listening to the current room, although story authors may wish to override it if a noisy object is nearby:

[listen]

• Diverts to [listen to $CurrentRoom].

Communication

A number of standard actions are related to communication. They divert according to a tree-like structure, eventually rooted in [talk to $] by default. But story authors may jack into this structure at any point.

[talk]

• Diverts to [talk to $], after asking for clarification.

[shout to $Obj]

• Diverts to [talk to $].

[shout]

• Diverts to [shout to $], after asking for clarification.

[call $Obj]

• Diverts to [shout to $].

[call]

• Diverts to [shout].

[ask $Obj about $Topic]

• Diverts to [talk to $ about $].

[tell $Obj about $Topic]

• Diverts to [talk to $ about $].

[talk to $Obj about $Topic]

• Diverts to [talk to $].

[ask $Obj]

• Diverts to [talk to $].

[tell $Obj]

• Diverts to [talk to $].

[greet $Obj]

• Diverts to [talk to $].

The input JEEVES, CLEAN MY SHOES is represented by the action [tell #jeeves to clean #shoes]. More generally:

[tell $Obj to | $Action]

• May divert to [greet $Obj] (if the inner action was [greet]).
• Otherwise, prints a stock message.

The ask and tell actions have a *topic parameter*. Topics can be ordinary objects, subject to normal scope rules. In addition, objects with the (topic $) trait are always recognized where the grammar expects a topic, even if they are out of scope—such objects can be used to represent abstract topics that do not correspond to tangible objects in the game world. Topics are described in more detail in the upcoming chapter on non-player characters.

Navigation

A number of standard actions are related to movement of the player character. These also form a tree of diversions, allowing actions to be intercepted at any level. However, when adjusting the behaviour of these actions, it is generally best to override one of the core actions, e.g. [leave $ $], [leave $ $ by $], or [leave $ $ with $].

[leave]

• Diverts to [leave $] or [go #out].

[stand]

• May divert to [leave $].
• Otherwise, prints a stock message.

[sit]

• Diverts to [climb $], after asking for clarification.

[push $Obj]

• Diverts to [push $Obj $], after asking for clarification.

[go $Dir]

• May divert to [leave $].
• Diverts to [leave $ $] or [leave $ $ by $].

[push $Obj $Dir]

• Diverts to [leave $ $ with $].

[go to $Room]

• Diverts to [go $] (multiple).

[find $Obj]

• Diverts to [go to $].

Miscellaneous actions

The verb USE is rarely used in interactive fiction, but it makes sense to treat it as a valid action for set phrases, such as USE TOILET, or LUKE, USE THE FORCE. The standard library accepts USE DOOR for entering doors. Furthermore, when the player has typed an incomplete command, such as CUT ROPE, the game might ask "With what?". In response to such a question, the player should be allowed to answer e.g. USE THE HERRING.

[use $Obj]

• May divert to any action, if the game recently asked for clarification.
• May divert to [enter $] (for doors).
• Otherwise, prints a stock message.

The following actions are so called *commands*. Their effects aren't part of the story, and no time passes in the game world when they are issued. Please refer to the library source code for details about what these actions do.

[notify off]

[notify on]

[pronouns]

[quit]

[restart]

[restore]

[save]

[score]

[transcript off]

[transcript on]

[undo]

[verbose]

[version]

AGAIN (G) and OOPS are treated as special cases by the parser. They are not actions.

Debugging actions

In addition to the standard library, the official Dialog distribution archive includes a *debugging extension*, which can be added to the compiler commandline before the standard library (but after the story). This extension adds a number of useful actions that generally should not be part of a released game. The in-game syntax should be evident from the action names:

[actions on]

Queries (actions on) to enable action logging. Whenever an action is about to be tried, its internal representation (data structure) and external representation (description) are printed.

[actions off]

Queries (actions off) to disable action logging.

[scope]

Queries (scope) to display the current scope.

[allrooms]

Prints a list of every room in the game, including unvisited ones.

[teleport $Room]

Moves the current player character into the named room.

[purloin $Object]

Moves the named object into the current player character's inventory.

[meminfo]

Prints a line of backend-specific memory statistics.

Onwards to “Chapter 8: Ticks, scenes, and progress” • Back to the Table of Contents