RFC - Dialogs specification

Metacosm Development Team

Revision History

Revision 1.5.1	April, 16th 2000	Revised by: Janselmeer
Small change in the 'expression modifiers' section		
Revision 1.4	April, 9th 2000	Revised by: Janselmeer
Improvement section 5.2, addition of notes about UI in sections 5.2 and 6.2, addition of expression modifiers		
Revision 1.3	March, 30th 2000	Revised by: Janselmeer
Conversion to metacosm-DTD		
Revision 1.2	November, 21th 1999	Revised by: Janselmeer
Addition of return codes and extension to dialogs between several creatures		
Revision 1.1	November, 17th 1999	Revised by: Janselmeer
Updates following discussions		
Revision 1.0	July, 22th 1999	Revised by: Janselmeer
First draft		

This document describes the dialogs (and monologues too).

1. License

2. Metacosm

For more information on Metacosm, check our site: Metacosm (http://metacosm.org).

3. Notations

Some terms are defined in the Vocabulary-RFC, which is available on the official web site.

Concerning dialogs case, keywords can be entered in lowercase, in uppercase or even in mixed case without any problem.

4. Bots Query Mode

4.1. Bases

Here the current subject is bots query mode specification. Query mode by keywords has been chosen. Each bot possesses a set of words, with which are associated answers.

Example 1. A bot whose name is Raoul and who is a baker

```
Name -> My name is Raoul.
Job  -> I am the baker.
```

Each bot will have to answer the following keywords:

- Name

- Job
- names of the creatures known by the bot
- names of locations known by the bot

Whereas the first two keywords are static, the following ones will be dynamic and will rely on bot memory. However, we have to nuance the staticity of the first two ones: a creature can lie or change job.

Example 2. Raoul is a baker who lives in North Midgaard

```
>Raoul
Bot: Raoul the baker is a vile enemy of mine.
     You can find it in the vicinity of North Midgaard.
>Marcel
Bot: I don't know that.
```

Keywords should also allow to move forward in a conversation:

Example 3. Keywords in conversation

```
>Job
Raoul: I am the baker.
>baker
Raoul: I cook breads, pies and cakes.
>cakes
Raoul: My cakes are the best in Midgaard.
```

But they can also be used as the thread of a quest:

Example 4. Keywords in a quest

```
>treasure
Old Pirate: I had a map but I sold it to an adven-
turer called Raoul.
(...)
>treasure
Raoul: I have only half of the map. Robert has the other part.
(...)
>treasure
Robert: I have the other part of the map.
```

4.2. Active Bots

Until now, we considered bots as completely passive, only answering players queries. But there is no reason and to the contrary bots should be more active in conversation.

They can propose discussion subjects (talk about the weather, local news), start a discussion with a passer-by (be it player or not), especially if the bot has something to sell.

Example 5. Meeting with a guard

A guard meet Janselmeer. He does not know who he is. So he is asking Janselmeer.

```
        A guard is approaching.
        Guard: What's your name, stranger?
case 1: >Janselmeer
        Guard: I am watching you, Janselmeer.
case 2: >Oumph
        Guard: I am watching you, Oumph.
```

Janselmeer can say his true name (1) or not (2). Both answers will be accepted by the guard. Janselmeer will be known until that moment under the name he gave (Janselmeer or Oumph) by this guard, and potentially by other people in town.

Now, let’s take this example again but the guard already knows the name of Janselmeer.

Example 6. Meeting with a guard (2)

```
        A guard is approaching.
        Guard: What’s your name, stranger?
case 1: >Janselmeer
        Guard: I am watching you, Janselmeer.
case 2: >Oumph
        Guard: You lie! I know that you are not Oumph.
        A guard attacks you.
```

Here, the guard waited ’Janselmeer’ as an answer. Any other answer, for instance ’Oumph’, can be interpreted as a lie by the guard.

This examples allows to touch two new aspects. First is the temporary addition of extra keywords. In the example above, it’s Janselmeer which has been added.

Second is the more interesting integration of player answers into the bot memory. In the examples, the guard will remember that the name of the player is Janselmeer. Of course, nothing prevents lying but the example shows incurred risks in a simplified way.

It can happen that one does not wish to answer questions. In this case, a creature can use the keyword JOKER to mean it.

The lack of an answer (after a limited time) could be considered as a refusal to communicate too.

Players will generally use the second solution and bots the first one. If players have to use JOKER, it would be better to alias it as "..." (or something else) in the UI, that is more explicit.

4.3. Communication between bots

Let us apply what we previously defined to communication between 2 bots. Let us go over the guard example again.

Example 7. Meeting with a guard (3)

As seen by the player, it should give the following:

```
Guard to Bot: What's your name, stranger?
Bot to Guard: My name is Bot.
Guard to Bot: I am watching you Bot.
```

As seen by a bot:

```
Guard to Bot: Name
Bot to Guard: Bot
Guard to Bot: ""
```

The main difficulty to achieve this is that the guard does not know that he talks to a bot. The one whom the guard talks to should be able to understand whether it is a bot or a player.

In order to make sure he is understood, the guard must therefore speak to his interlocutor in a way understandable by players and non-players alike. Moreover, if there are other creatures who listen to the conversation, they will hence be able to understand too, whether they be bots or players.

The solution that the example suggests is to integrate into a message both the short version interpretable by bots (corresponding to keywords), as well as a long version for players.

Example 8. Meeting with a guard (4)

As seen by the game:

```
Guard to Bot: ("Name", "What’s your name, stranger?")
Bot to Guard: ("Bot", "My name is Bot.")
Guard to Bot: ("", "I am watching you Bot.")
```

When a bot or a player sends a message which can not be understood or which does not need an answer, the short version will be empty. No processing will then be done in order to interpret the message. On the other hand, it could be repeated.

Example 9. Meeting with a guard (5)

As seen by the game:

```
Guard: Now you die, Raoul!
Message ("", "Now you die, Raoul!")
```

Example 10. The grocer

```
        Grocer: Look at my beautiful waterskins! Buy them!
(a priori destined only to players, unless bots are uncan-
nily clever)
        Message ("", "Look at my beautiful waterskins! Buy them!")
```

4.4. Application to user interface level

The player has two ways of sending messages:

- classical

  Example 11. Where was Oumph?

tell Oumph "Where have you been?"

This will generate this message ("", "Where have you been?"). It could not be understood by a non-player; however every creature is able to report the words that she hears (spy, gossip, etc).

- with aliases

  Example 12. Oumph's job

  This will generate this message ("Job", "What's your job?") which can be understood by bots and players.

  An alias list will have to be implemented in the controller. The player will of course be able to modify them in his own way.

  If the alias does not exist, 'tell Oumph cloporte' will generate the message ("cloporte", "cloporte").

4.5. Control codes

In order that the bots could understand one another better, we add return codes NONE, ANS, WRONG_ANS and BAD_ANS. These codes are not readable by players and cannot be used directly by players either (the 'answer' commands will allow to send a message with the control code ANS). So the codes using is reserved to bots.

NONE is the default control code. It does not mean anything. It is employed when no other control code is used.

The code ANS means that the message is an answer to a previous question.

WRONG_ANS is used to point out that the answer is regarded as wrong and deceitful. The interlocutor can therefore give another chance to say the truth (or rather what the bot considers as the truth), change topic of conversation or break the discussion with troublesome consequences (fight, reputation drop...).

Example 13. A guard is questioning a player

```
A guard is approaching.
Guard: What’s your name, stranger?
(name, "What’s your name, stranger?", NONE)
>Janselmeer
("Janselmeer", "Janselmeer", ANS)
Guard: You lie! I know you are not Janselmeer.
("", "You lie! I know you are not Janselmeer.", WRONG_ANS)
>Oumph
("Oumph", "Oumph", ANS)
case 1:
  Guard: You lie! I know that you are not Oumph.
  ("", "You lie! I know that you are not Oumph.", WRONG_ANS)
  A guard attacks you.
case 2:
  Guard: I’m watching you Oumph.
  ("", "I’m watching you Oumph.", NONE)
```

It is advisable to limit the number of possible tries... BAD_ANS marks that the answer is incorrect, inconsistent with the answer type which was expected.

Example 14. A guard is approaching

```
A guard is approaching.
Guard: What’s your name, stranger?
(name, "What’s your name, stranger?", NONE)
>43
("43", "43", ANS)
Guard: "I said ’What’s your name?’."
("", "I said ’What’s your name?’.", BAD_ANS)
```

A BAD_ANS code generated by a bot is a serious error in the game code or in the bot code. An error log must be made for debugging and correction.

A BAD_ANS code generated by a player means a typo, a bad understanding of the question, a voluntary lie, a try at disturbing the bot, etc.

The coder is free to make its bot react as she thinks best (polite and understanding, nervous and aggressive, etc). So the bot can repeat its question or not.

the bot cannot and should not know whether it speaks to another bot or to a player. This is the reason why the number of BAD_ANS answers must be limited, in order to prevent buggy bots to induce infinite loops...

4.6. Expression modifiers

An Expression Modifier is a context information going with the message. This information is relevant for the bot receiving the message and help him decide how to react.

For example, there are several ways to ask somebody its name:

- polite: "Would you mind to tell me your name, please?"
- neutral: "What is your name?"
- insulting: "What is your name, alien scum?"

Each time, only the expression modifier changed. Below are the same messages understood by bots:

- polite: (name, NONE, POLITE)
- neutral: (name, NONE, NEUTRAL)
- insulting: (name, NONE, INSULTING)

POLITE, NEUTRAL and INSULTING are Expression Modifiers. Here is a list of expression MODIFIERS:

- OBSEQUIOUS
- POLITE
- NEUTRAL
- RUDE
- THREATENING
- INSULTING
- CHARMING

This list is not closed.

The following example explains how Expression Modifiers could be used. Janselmeer meets Robert the butcher (Raoul's brother) and asks him about Raoul the baker.

Example 15. Expression modifiers

```
Janselmeer: (RAOUL, "Do you know RAOUL?", NONE, NEUTRAL)
        Robert:  ("", "I am not con-
cerned with that RAOUL.", ANS, RUDE)
Janselmeer: (RAOUL, "O MIGHTY ONE, may I beg you a little of your
                     knowledge about Raoul?", NONE, OBSEQUIOUS)
        Robert: ("", "RAOUL? This thing ex-
ists only in your sick mind.",
                 ANS, INSULTING)
Janselmeer: (RAOUL, "Tell me some-
thing about Raoul or I'll kill you!",
                     NONE, THREATENING)
        Robert: ("", "I swear I don't know Raoul.", ANS, NEUTRAL)
        Janselmeer punches at Robert.
Janselmeer: (RAOUL, "Tell me some-
thing about Raoul or I'll kill you!",
                     NONE, THREATENING)
        Robert: ("", "Raoul is the baker. He lives North of here. Please don 't
```

```
kill me!", ANS, POLITE)
```

There is no obligation using an Expression Modifier whether another. It is a pure matter of roleplay. Some people are used to be rude and some others are used to be polite only because it is their way. But of course creatures may not have the same reaction with the first than with the second. It depends on them. For example, a lord is normally used to be talked obsequiously and may be offended by someone that would act differently. So the choice of an expression modifier depends on both the personality of the character that is speaking and the expected reaction of the character that is listening.

4.7. Future extensions of bots language

The keywords system has been chosen on grounds of simplicity of implementation. Nevertheless, this should evolve and lead towards a true dialog with complete sentences (subject, verb, complement).

The first extension to keywords could be the use of interrogative pronouns and adverbs: what, who, where, why, how, how many, which, when.

Example 16. Pronouns

```
>Who Raoul
Bot: Raoul is the baker.
>Where Raoul
Bot: You can find it in the vicinity of North Midgaard.
>What Raoul
Bot: I've never heard of this.
```

5. The various modes of verbal communication

The content of this part can seem obvious but yet it is altogether essential. We distinguish two communication modes which are dialog and monologue, which will be used to different ends in the game.

5.1. The monologue

What we call monologue is when a creature sends a self-supporting message and hence does not wait for an answer.

This message can be intended for a creature in particular, but most of the time it is not specifically aimed.

Example 17. A guard attacks Janselmeer

```
Guard: Now you die, Janselmeer!
A guard attacks Janselmeer.
```

This message is aimed merely to Janselmeer, but all creatures passing by can hear it.

Example 18. A caravan arrives to the market place

```
The caravan: The caravan has arrived! Look at my beauti-
ful vegetables!
```

This message is intended for all creatures passing by.

Example 19. Janselmeer has just bought something

```
Janselmeer: That was my last silver...
```

The message is aimed to no one but can be heard by all present creatures (who will wonder about Janselmeer's health :).

5.2. The dialog

Dialog

> The dialog is an information exchange by means of messages between two or more creatures.

It is carried out in connected mode, with a connection phase, a conversation phase and a disconnection phase.

Let us see an instance of dialog: a tourist who asks his way to a passer-by.

```
        -connection phase:
tourist: Please, sir...
The passer-by stops
passer-by: Yes?

-conversation phase:
tourist: I'm looking for the local camp site.
passer-
by: You turn to the left 500m from here, then go straight ahead
           to the fountain and then turn to the right.

        -disconnection phase:
tourist: Thank you, have a nice day!
passer-by: You're welcome.
```

It's this kind of dialog which we will attempt to reproduce. A dialog unfolds in the following way:

```
    The C1 creature wants to talk to creatures C2, C3, ..., Cn.
    1- C1 asks to talk to C2, C3, ..., Cn.
    2- Some Ci accept the request, others decline it.
    3- C1 talks with the ones who have accepted.
    4- Those taking part in the conversation withdraw gradu-
ally. The last but one
    leave will put an end to the conversation.
```

The advantage of a connected mode is to make the control of bots easier. For example, if one speaks to a bot which is moving, this bot will stop during the conversation (hence during the connection).

The dialog should then be the privileged communication mode with bots. As soon as a dialog is opened, some actions are forbidden (as going away for instance). These actions automatically break the conversation.

A dialog can be interrupted by an external event only if this one requires a mandatory reaction from a creature involved in the conversation. Such an event can be a fight for example.

```
           Tourist                        |             Passer-by
        -----------------------------------------------------------
------------
>talk passer-by                  |
Tourist: Please sir...           | Tourist: Please sir...
                                 | A tourist wants to talk with you.
                                         | Do you accept? (Y/N)
| >Y
        passer-by: Yes?... | passer-by: Yes?...
>tell passer-by "I'm looking     |
 for the local camp site."       | tourist: I'm look-
ing for the local
                                         | camp site.
                                 | >tell tourist "You turn to the left
```

```
                                          |  500m from here, then go straight
                                          |  ahead to the foun-
tain and then turn
                                          | to the right."
passer-by: You turn to the left |
  500m from here, then go on    |
  straight ahead to the         |
  fountain and then turn to the |
  right.                        |
tell passer-by bye |
Tourist: Thank you, have a nice |
  day!         | tourist: Thank you, have a nice day!
```

This is worth noting that the command 'talk' allows dialog opening and that the keyword 'bye' puts an end to the conversation.

Note too, that an UI can provide some shortcuts in the dialog. for example, it is not needed to ask the players if they want to answer. It is possible just let them answer if yes and do something else if no. In a similar fashion, to break the dialog, it should be enough just to do something else and not to have to type bye.

In order to open a dialog with more than one creature, one adds the names of the desired creatures in the parameters to 'talk', or one uses 'talk~all' in order to talk to all present creatures.

Example 20. Hiring in a tavern

```
        Recruiter> talk all
        Recruiter: "I beg your attention, please."
        [To this stage, some listen, some others don't.]
        Recruiter: "I've got a job to offer, danger-
ous and underpaid."
        [Two or three creatures are still listening...]
        Recruiter: "3 silvers to go kill Skarkash."
        [Huge silence in the room]
        Recruiter: "Well, never mind!"
        Recruiter> bye
```

```
Recruiter: "Thanks for your attention."
```

Example 21. Identity control in the street

The guard is searching for an escaped elf prisoner.

```
Guard> talk elf1 elf2
Guard: "Hey you, and you!"
Guard: "Give me your names!"
Elf1: "Celeborn"
Elf2: "Galadriel"
Guard> bye
Guard: "All right, move along."
```

this example shows the underlying coding difficulties. The sentence associated to a keyword depends on the language level of the one who speaks (politeness, elocution, etc), on its work, on the number of persons to which one speaks to, on the sex of these persons, etc.

5.3. Messages and perception

The dialog system depends on perceptions. A message is indeed nothing but a special stimulus which holds a lot of information. A message is associated to one or several stimuli, which makes it possible to choose the medium (sound, telepathy, etc) and the intensity of the message. So it is possible to choose one's means to communicate.

6. Multilingual dialogs

6.1. Various languages in the game

Creatures in a given world do not have necessarily the same language nor the same way to express themselves.

In order to manage the differences between languages, we introduce a ’language’ attribute in messages.

On receiving a message, a creature will have to verify that it knows the language in which this message is formulated, before being able to read it. If this is not the case, the message text will appear encrypted (and so unintelligible) to the player or to the script/AI which receives it.

A creature can anyway memorize a message which it do not understand (language problem). To the contrary, it cannot memorize a message if it does not possess the sense required for the perception of the message.

6.2. Multilingual support for players

In order to face the objective of a multilingual game (English, French, Spanish, German...), it is required that dialogs be the same as much as we can.

The solution would consist in integrating in the messages the same version of the text in various languages.

It would concern messages sent by bots but also the texts of player aliases.

Example 22. Multilingual example

```
job -> ("What’s your name?", "Comment t’appelles-tu~?", "Wie heisst du~?")
```

Therefore players could read messages from the desired version if this version is available, otherwise it will be the default version (which is always available). On the other hand, players will of course not have to formulate their messages in several languages.

The internal representation of standard keywords (name, job...) will always stay the same but it is not out of the question that these keywords be translated by the interface. Other keywords will definitely have to be translated in several languages by world creators or translators.

The following example shows well the necessity of translating some keywords:

Example 23. Example of the baker

in English

```
>Job
Raoul: I am the baker.
>baker
Raoul: I cook breads, pies and cakes.
>cakes
Raoul: My cakes are the best in Midgaard.
```

In French

```
>Job
Raoul: Je suis le boulanger.
>boulanger
Raoul: Je fais du pain, des tartes et des gâteaux.
>gateaux
Raoul: Mes gâteaux sont les meilleurs de Midgaard.
```

If we do not translate the keywords

`baker`

and

`cakes`

, a player in French version will have some trouble to advance in the conversation:

Example 24. Bad translation

```
>Job
Raoul: Je suis le boulanger.
>boulanger
Raoul: Je ne connais pas.
>gateaux.
Raoul: Je ne connais pas.
```

This is a trap into which translators must not fall.

Finally, we specify to avoid any misunderstanding, that a keyword in French does not mean that the answer will be merely in French, and ditto for other languages. In all events, the answer must remain multilingual so that the other players who listen to the message and who use another language, may understand. Actually, there will be only a set of keywords associated to a set of answers:

```
    (boulanger, baker)-
>("Je fais du pain, des tartes et des gâteaux.",
    "I cook breads, pies and cakes.")
```

7. Summary

Standard keywords (valid for all bots):

- job = "What is your job?": request of the activity of the creature.

- name = "What is your name?": request of the name of the creature.
- joker = refusal of answering a question.

Control codes:

- NONE: default (no signification).
- ANS: answer.
- BAD_ANS: not a valid answer.
- WRONG_ANS: wrong answer.

Expression modifiers:

- OBSEQUIOUS
- POLITE
- NEUTRAL
- RUDE
- THREATENING
- INSULTING
- CHARMING

A messages consists in:

- an attribute giving the language,
- an attribute giving the addressee(s) of the message,
- a short text interpretable by bots,
- a long text for players available in several languages (including at least a default language),

- a control code,
- an expression modifier.

Commands:

- tell: sends a message to one or several entities.
- talk: opens a dialog with one or several entities.
- bye: closes a dialog.
- answer: answers to a question.