TADS 3.0.9
DYNAMIC CHARACTERS QUICK REFERENCE

ActorState

- **stateDesc** - this is a message that's added to the actor's basic description (the "npcDesc" property in the Actor object). Most actors will have a permanent description that never changes - a basic description of their physical appearance - along with some extra information that describes what they're doing right now. The stateDesc lets you add this extra state-dependent part.
- **specialDesc** - displays the actor's in-room description. This is the description displayed in the room description (for example, when entering the room, or in response to a LOOK command). By default, we'll invoke the actor's actorHereDesc method.
- **distantSpecialDesc** - the special description of the actor from a distance.
- **remoteSpecialDesc(actor)** - the special description of the actor from a remote location.
- **obeyCommand(issuingActor, action)** - determine if we should obey the given action. By default, we'll simply refuse all commands and display any matching CommandTopic.
- **greetingsFrom(actor)** - receive greetings from the given actor; this is called when we're the target of a command like "bob, hello" or "talk to bob." By default, this simply says "there's no response."
- **goodbyeFrom(actor)** - this is called when we're the target of "actor, bye." The default is to say "there's no response."
- **yesFrom(actor), noFrom(actor)** - these are called when we're the target of a command like "bob, yes" or "bob, no." The default is to say "there's no response."
- **takeTurn()** - this is called once per turn. This allows the actor to carry out scripted behaviour appropriate to the current state. This does nothing by default.
- **afterAction() -** this method can contain code to be run after the PC performs an action in the NPC’s scope.
- **beforeAction() -** this method can contain code to be run before the PC performs an action in the NPC’s scope; the action can be vetoed with an exit command.
- **beforeTravel(traveler, connector)**
- **afterTravel(traveler, connector) -** these methods are invoked just before and after a traveler travels via traveler-TravelTo(); beforeTravel() is called on each object connected by containment to the traveler in its old location, and afterTravel() is called on each object connected by containment in the new location. These notifications are more precise than using beforeAction() and afterAction() with the TravelVia pseudo-action, because these actions are only called when travel is actually occurring. TravelVia will fire notifications even when travel isn't actually possible. A beforeTravel() method can veto the travel action using "exit". The notification is invoked before the travel is actually performed, and even before a description of the departure is produced.
- **isInitState** - if set to true, then the then the library will automatically set the corresponding actor's curState property to point to that ActorState during pre-initialization.
- **arrivingTurn() -** When group travel is performed using the AccompanyingInTravelState class, this is essentially called in lieu of the regular takeTurn() method on the state that is coming into effect after the group travel.
- **autoSuggest** - set to nil to disable automatic topic inventory listing on TALK TO commands.

HermitActorState

- **noResponse** - explain why actor in this state is too busy or otherwise unresponsive.

Accompanying State

- **accompanyTravel(leadActor, conn)** { return leadActor == gPlayerChar; } to make NPC follow player.
- **getAccompanyingTravelState(leadActor, conn)** { return new WhoeverTravelState(location, leadActor, self); }

AccompanyingInTravelState

- **sayDeparting(conn)** - the ‘before’ message
- **specialDesc** - the ‘after’ message

ConversationReadyState

- **inConvState** - our in-conversation state object. This specifies the state to switch to when a conversation begins. This should be set to an **InConversationState** object. For your convenience, rather than defining this property explicitly, you can put the ConversationReadyState "inside" its corresponding InConversationState, using the '+' syntax. If a ConversationReadyState is nested within an InConversationState object, the library will automatically initialize the former's inConvState property to point to the containing state.
- **Greeting Protocols** - use HelloTopic, ImpHelloTopic, ByeTopic and ImpByeTopic, located in the ConversationReadyState. The ImpXXXTopic forms are for implicit greeting or parting; if absent, the XXXTopic entry will be used instead.

InConversationState

- **attentionSpan** - this is an integer giving the number of turns the actor should wait before giving up on the conversation. The default is 4. If the other character doesn't talk to our NPC for this many turns, we'll automatically terminate the conversation, switching to our next state. If the NPC’s attention span is unlimited, set this to nil.
- **nextState** - this is an ActorState object, which should usually be of the ConversationReadyState subclass, which follows the conversation's termination. When we terminate the conversation, we'll switch to this state. You don't have to override this; if you don't, we'll remember the state that the actor was in just before the conversation, and switch back to that state when the conversation ends.

Actor

- **initiateConversation(state, node)** - see ConvNode
- **initiateTopic(obj)** - used with InitiateTopic (q.v.)

TopicEntry *Subclasses*

- **AskTopic** answers an ASK ABOUT question;
- **TellTopic** responds to a TELL ABOUT command;
- **AskTellTopic** answers either an ASK ABOUT or TELL ABOUT command;
- **GiveTopic** responds to a GIVE TO command;
- **ShowTopic** responds to a SHOW TO command;
- **GiveShowTopic** responds to either a GIVE TO or SHOW TO command; **AskTellGiveShowTopic**.
- **AskForTopic** responds to an ASK FOR command
- **AskAboutForTopic, AskTellAboutForTopic**
- **YesTopic** responds to a YES command.
- **NoTopic** responds to a NO command.
- **ByeTopic & ImpByeTopic & HelloGoodbyeTopic**
- **HelloTopic & ImpHelloTopic** - use these five on a ConversationReadyState to provide greeting protocols (although you can also include HelloTopic and ByeTopic on other ActorStates to handle these commands.)
- **CommandTopic** - provide a response to a command issued to the actor, e.g. “Bob, read the book” would activate CommandTopic @ReadAction.
- **InitiateTopic** responds to Actor.initiateTopic(obj)

TopicEntry *Methods & Properties*

- **matchObj** - My matching simulation object or objects. This can be either a single object or a list of objects.
- **getActor()** - The Actor object to whom this topic ultimately belongs
- **isActive** - the condition that determines when the topic entry should become active. We'll never show the topic's

response when isActive returns nil. Allows conditional responsibilities, especially in conjunction with **AltTopic** and TopicGroups.

- **isConversational** - true by default, if nil does not trigger greeting protocols.
- **matchScore** - the match strength score. By default, we'll use a score of 100, which is just an arbitrary base score.
- **topicResponse** - Our response. This is displayed when we're the topic entry selected to handle an ASK or TELL. Each topic entry must overridethis to show our response text (or, alternatively, an entry can override handleTopic(fromActor, topic) so that it doesn't call this property)
- **matchPattern** - match a regular expression pattern (as an alternative to matchObj).

TopicEntry Templates

- TopicEntry template +matchScore? @matchObj | [matchObj] | 'matchPattern' "topicResponse" | [eventList] ?;
- TopicEntry template +matchScore? @matchObj | [matchObj] | 'matchPattern' [firstEvents] [eventList];
- TopicEntry template +matchScore? @matchObj | [matchObj] 'matchPattern' "topicResponse" | [eventList] ?;
- TopicEntry template +matchScore? @matchObj | [matchObj] 'matchPattern' [firstEvents] [eventList];

- MiscTopic template "topicResponse" | [eventList];
- MiscTopic template [firstEvents] [eventList];

DefaultTopics

- DefaultAskTopic, DefaultTellTopic, DefaultAskTellTopic, DefaultGiveTopic, DefaultShowTopic, DefaultGiveShowTopic, DefaultAskForTopic, DefaultInitiateTopic, DefaultCommandTopic, DefaultAnyTopic

AltTopic

- **isActive** - the condition that must be true for this topic to be used instead of the TopicEntry within which it is directly nested.

TopicGroup (nest inside the appropriate ActorState; nest related TopicEntry or TopicGroup objects within the TopicGroup)

- **isActive** - the condition that allows this topic group to be active.

TopicInventory

There are three ways the topic inventory can be displayed:

- in response to a TOPICS command from the player;
- in response to a TALK TO command from the player;
- any other time the game (or library) thinks it's a good idea.

SuggestedTopic Subclasses:

- SuggestedAskTopic, SuggestedTellTopic, SuggestedShowTopic, SuggestedGiveTopic, SuggestedYesTopic, SuggestedAskForTopic and SuggestedNoTopic.

Properties

- **name** - this is the name that will be shown in the suggested topic list. It should be given so that it can be substituted into a sentence after "ask about" (or the appropriate variation for the other types), so it should usually include "the" if appropriate.

ConvNode (Nest directly inside Actor; nest topic entries in ConvNode)

- + ConvNode 'name'
- **npcGreetingMsg** - use this to display a message when the NPC initiates a conversation, generally via a call to **Actor.initiateConversation(ActorState, 'name')**.
- **npcGreetingList** - use as an alternative to npcGreetingMsg in cases where the ConvNode may be initiated more than once.
- **npcContinueMsg** or **npcContinueList** - the InConversationState class automatically displays the current ConvNode's continuation message (using either npcContinueMsg or npcContinueList, as appropriate) on each turn on which the ConvNode is active, and the player didn't address a conversational command to the NPC on the same turn.
- **endConversation(actor, reason)** - Instances can override this for special behaviour on terminating a conversation.
- **canEndConversation(actor, reason)** - lets a node prevent a conversation ending, by returning nil; **reason** can be endConvBye, endConvTravel, or endConvBoredom.
- **isSticky** - if this flag is true (it's nil by default) then this conversation node remains current until a response explicitly changes the node.

SpecialTopic (only for use in ConvNode)

- *example*: ++ SpecialTopic 'call him a liar' ['call', 'bob', 'him', 'a', 'liar']
  "<q>You're so full of crap, Bob [...] " ;

Tags (for use in TopicEntry & SpecialTopic object text response output)

- **<.convode name>** - switches the NPC that issued the response to the named conversation node. You can also set the tree position explicitly by calling the NPC's setConvNode(name) method, passing the new conversation node name as the parameter.
- **<.convstay>** - - keep the responding actor in the same conversation node as it was in at the start of the current response
- **<.topics>** -schedule a topic inventory for the end of the turn.
- **<.reveal key> -** add 'key' to the knowledge token lookup table. The 'key' is an arbitrary string, which we can look up in the table to determine if the key has even been revealed. You can refer to that lookup table using gRevealed('key'), which returns true if the 'key' value has been revealed using the <.reveal> tag, nil if not. Use the macro gReveal('key') to explicitly add the key to the table of revealed keys outside text output.

EventList Classes (use as mix-ins with TopicEntry objects)

- StopEventtList, RandomEventList, ShuffledEventList, SyncEventList, CyclicEventList, ExternalEventList

ShuffledEventList

- **firstEvents** - can be set to a list of strings to show sequentially before starting the shuffled strings.
- **shuffleFirst** - set this to nil (it's true by default) if you don't want the list shuffled the first time through

SyncEventList

- **masterObject** - the EventList object (e.g. bobSweeping. greetingList) with which this list is to be kept in sync; both lists will advance their positions in unison.

AgendaItem

- **agendaOrder** - the ordering of this AgendaItem relative to other AgendaItems. When two or more AgendaItems are ready in the same turn, the AgendaItem with the lowest agendaOrder will be selected for execution.
- **initiallyActive** - set this to true to make this AgendaItem active (added to the Actor's agendaList) at the beginning of the game.
- **isDone** - set to true when this AgendaItem is finished with and can be removed from its actor's agendaList.
- **isReady** - set to true when this AgendaItem is ready to be invoked (note it must also be in its actor's agendaList for invocation to occur).
- **invokeItem()** - Execute this item. This is invoked during the actor's turn when the item is the first item that's ready to execute in the actor's agenda list.
- To add this item to the actor's current agenda, call **addToAgenda(item)** on the *actor*.