Author: *Master*

ARMA 2 CONVERSATION SYSTEM

The ***conversation system*** in ArmA 2 has replaced the **description.ext** approach.
Main advantages:

- The system reacts on the sentence to start.
- Conversation flows through the channel:
    - direct for *unit-to-unit*
    - via radio for team members etc.
- Possibility to create dynamic conversations.
- FSM syntax allows editing.
- The same ***conversation topic*** has to be added to each of the participants.
- The conversation starts with the sentence which one of the participants says to another.
- The conversation can be controlled by **SQF**-scripts: **FSMs** and ***event handlers*** assigned to each participant.
- After a participant receives a sentence, its script reacts to its ID.

Adding the ***conversation topic*** to a participant:

```
participant kbAddTopic [“topic name”, “topic config”, “FSM”, “event handler”]
```

Where:

- topic name - the title (String)
- topic config - the topic configuration (**.bikb**) (String)
- FSM - the *Finite State Machine* (**.fsm**) (String) – for unit controlled by AI
- event handler - the script (**.sqf**) (String or code) - for unit controlled by a player

Bohemia Interactive Knowledge Base (**.bikb**) file was used to store an AI unit's memory of what it has seen. Now the file is used to store the text and sound samples of the sentences for the ***conversation topic***.
NOTE: This is similar to the ***class*** **cfgSounds** and ***class*** **cfgRadio** in **description.ext**.
Example:

```
class Sentences {
	class say1 {
		text = "Hello unit1.";
	speech[] = {"\Sound\unit2_01.ogg"};
		class Arguments {};
	};
	class say2 {
		text = "Hello unit2.";
		speech[] = {"\Sound\unit1_01.ogg"};
		class Arguments {};
	}
};
class Arguments{};
class Special{};
startWithVocal[] = {hour};
startWithConsonant[] = {europe, university}
```

Where:

- *text* - the text sample of a sentence.
  NOTE:
  - Using stringtable.xml, type **"$STR_ClassName"** (with the *dollar* sign ($) followed by capitalized prefix **STR_**).
  - Place your **stringtable.xml** of the mission folder. If simulation engine cannot find the string ID in the string table of the mission then it looks in the core string table.
- *speech* – the sound sample of a sentence.
  NOTE: You can use the sentences without the sound: **speech[] = {""}**.
- *Arguments* and *Special **classes***, you can safely ignore these.
- *startWithVocal* and *startWithConsonant* arrays, you can safely ignore these.

NOTE:

- All dubbing sound files are packed in **dubbing.pbo** addon.
- Sentences are defined in ***.bikb** files located in subfolder **\kb** within a mission folder.
- FSM is executed every time an AI unit receives a sentence, so you need to check for _sentenceID in separate conditions right at the start.
- You can open as many conversation menus as you want in player's event-handler code (most commonly an ***.sqf** file). In the FSM, you are just checking for the interrupted event. If more of those can occur in one topic, you can check which sentence was said last via **kbWasSaid** command.
- **kbTell** command always works only locally

Class *Interrupted*

The quick action menu on the HUD can be closed via *Backspace* key. If you want to handle this event, you have to add new ***class Interrupted*** into the ***class Sentences***.

```
class Sentences {
	// Other classes
	class Interrupted {
		text = "";
		speech[] = {""};
		class Arguments {};
	};
};
```

It can be used as a default _*sentenceId* in the script.

Scripts

The **FSM** and ***event handler*** parameters are optional in **kbAddTopic** definition.
While the unit receives a sentence, the engine defines who the unit controller is:

- If it is controlled by AI, the **FSM** is executed.
- If it is controlled by player, the ***event handler*** is executed.

NOTE: If you are making a multiplayer mission and the unit is playable, you will want to use both the **FSM** and the ***event handler*** together.
The default variables of ***FSMs*** and ***event handlers***:

- _*this* – the receiver of the sentence.
- _*from* – the sender of the sentence.
- _*sentenceId* – the ***class*** of the sentence that the receiver is reacting to.
- _*topic* - the topic name of the conversation.

FSM

FSM stand for *Finite-State Machine* system. **FSM** is executed only once after each received sentence.

Event Handler

Event handler is executed:

- if it receives a sentence
- if the player points at an interlocutor and is close enough to start a conversation.

NOTE: This is not standard event handler.

NOTE: You cannot use either ***sleep*** or ***waitUntil*** command in the **FSM** as they are precompiled.

If you need to add a delay into the code, you have to start a new script scope via ***spawn*** command.

NOTE: You can use the default variable **_topic** instead of the topic name (String) in the ***.fsm** and ***.sgf** files of the topic. Use the topic name (String) in the other scripts.

Non-FSM control

You can control a conversation via any script without **FSM** and ***event handler***.

Example:

```
unit1 kbAddTopic ["dialog", "kb\ dialog.bikb"];
unit2 kbAddTopic ["dialog", "kb\ dialog.bikb"];

unit1 kbTell [unit2, "dialog", "dialog_u1_1"];
waitUntil { unit1 kbWasSaid [unit2, "dialog", "dialog_u1_1", 3]};

unit2kbTell [unit1, "dialog", " dialog_u2_1"];
waitUntil {unit2 kbWasSaid [unit1, "dialog", "dialog_u2_1", 3]};

//Conversation ended.
```

Creating Conversation

In the mission editor:

1. Create a character controlled by a player and call it as “unit1”.
2. Create other character and call it as “unit2”.
3. Save mission, go to and open the mission folder.

In the mission folder:

4. Create the file **stringtable.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<Project name="Arma2">
	<Package name="Missions">
		<Container name="Conversations">
			<Container name="ConvActMenu">
				<Key ID="conv_actmenu_say">
					<English>Say</English>
				</Key>
				<Key ID="conv_actmenu_silent">
					<English>Keep silent</English>
				</Key>
				<Key ID="conv_actmenu_continue">
					<English>Continue</English>
				</Key>
				<Key ID="conv_actmenu_discontinue">
					<English>Discontinue</English>
				</Key>
				<Key ID="conv_actmenu_yes">
					<English>Yes</English>
				</Key>
				<Key ID="conv_actmenu_no">
					<English>No</English>
				</Key>
			</Container>
			<Container name="Dialog">
				<Key ID="str_conv_unit1_sent_1">
					<English>Hi! How are you?</English>
				</Key>
				<Key ID="str_conv_unit2_sent_2">
					<English>Hi! All right!</English>
				</Key>
				<Key ID="str_conv_unit1_sent_3">
					<English>You'll watch football today?</English>
				</Key>
				<Key ID="str_conv_unit2_sent_4">
					<English>Yes! And you?</English>
				</Key>
				<Key ID="str_conv_unit1_alter_yes">
					<English>Yes, I'll watch football. Goodbye.</English>
				</Key>
				<Key ID="str_conv_unit1_alter_no">
					<English>No, I will not watch football. Goodbye.</English>
				</Key>
				<Key ID="str_conv_unit2_confirm">
					<English>Good luck to your team. Goodbye.</English>
				</Key>
				<Key ID="str_conv_unit2_cancel">
					<English>Goodbye.</English>
				</Key>
			</Container>
		</Container>
	</Package>
</Project>
```

NOTE: You can only use languages supported by a product.

To see the languages supported:

- Refer to official game site
- Unpack the file **languages.pbo** and open **stringtable.xml** (as example).

5. Create the folder “kb”.

In the folder **kb**:

6. Create the file **conv.bikb**:

```
class Sentences {
	class conv_unit1_sent_1 {
		text = "$STR_conv_unit1_sent_1";
		speech[] = {""};
		class Arguments {};
	};
	class conv_unit2_sent_2 {
		text = "$STR_conv_unit2_sent_2";
		speech[] = {""};
		class Arguments {};
	};
	class conv_unit1_sent_3 {
		text = "$STR_conv_unit1_sent_3";
		speech[] = {""};
		class Arguments {};
	};
	class conv_unit2_sent_4 {
		text = "$STR_conv_unit2_sent_4";
		speech[] = {""};
		class Arguments {};
	};
	class conv_unit1_alter_yes {
		text = "$STR_conv_unit1_alter_yes";
		speech[] = {""};
		class Arguments {};
	};
	class conv_unit1_alter_no {
		text = "$STR_conv_unit1_alter_no";
		speech[] = {""};
		class Arguments {};
	};
	class conv_unit2_confirm {
		text = "$STR_conv_unit2_confirm";
		speech[] = {""};
		class Arguments {};
	};
	class conv_unit2_cancel {
		text = "$STR_conv_unit2_cancel";
		speech[] = {""};
		class Arguments {};
	};
	class Interrupted {
		text = "";
		speech[] = {""};
		class Arguments {};
	};
};

class Arguments{};
class Special{};
startWithVocal[] = {hour};
startWithConsonant[] = {europe, university};
```

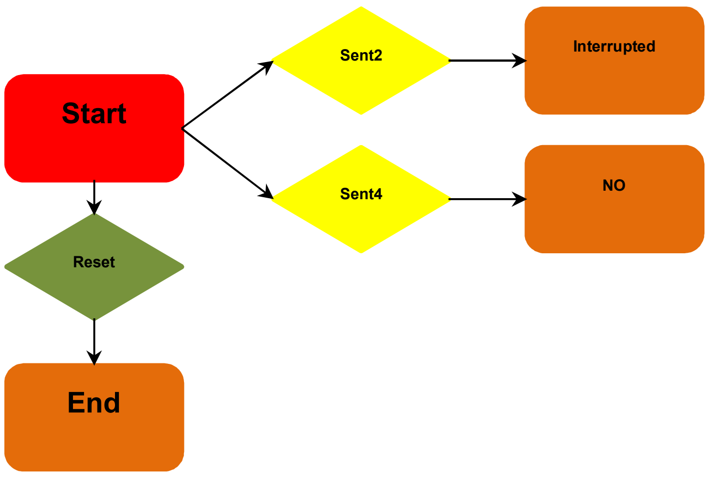
7. Create the file **conv_unit1.sqf** (for **unit1** controlled by a player):

```
// Collect actions (if any) to the quick action menu via the BIS_convMenu array.
// Parameters : <menu_item> (String), _topic (String), _sentenceid (String).
BIS_convMenu = [];

// Check: if the player is pointing at the interlocutor
// Check: if the player says to the interlocutor
// Check: if the player said to the interlocutor
if (_from == VIED_Hooker1 && _sentenceId == "" && !(_this kbWasSaid [_from, _topic, "conv_unit1_sent_1", 120])) then
{
		// Add the sentence to BIS_convMenu for the player
		BIS_convMenu = BIS_convMenu + [[localize "conv_actmenu_say", _topic, "conv_unit1_sent_1", []]];
		BIS_convMenu = BIS_convMenu + [[localize "conv_actmenu_silent", _topic, "Interrupted", []]];
};

switch (_sentenceId) do
{
		case "conv_unit2_sent_2":
		{
				BIS_convMenu = BIS_convMenu + [[localize " conv_actmenu_continue", _topic, "conv_unit1_sent_3", 
[]]];
				BIS_convMenu = BIS_convMenu + [[localize " conv_actmenu_discontinue ", _topic, "Interrupted", []]];
		};
		case "conv_unit2_sent_4":
		{
				BIS_convMenu = BIS_convMenu + [[localize "conv_actmenu_yes", _topic, "conv_unit1_alter_yes", []]];
				BIS_convMenu = BIS_convMenu + [[localize "conv_actmenu_no", _topic, "conv_unit1_alter_no", []]];
		};

		default {};
};
// Return the result to the scope
BIS_convMenu;
```

8. Create the file **conv_unit2.sqf** (for **unit2** controlled by an ***artificial intelligence* (AI)**):

```
// Collect actions (if any) to the quick action menu via the BIS_convMenu array.
// Parameters : <menu_item> (String), _topic (String), _sentenceid (String).
BIS_convMenu = [];

switch (_sentenceId) do
{
	case "conv_unit1_sent_1":
	{
		_this kbtell [_from, _topic,"conv_unit2_sent_2"];
	};

	case "conv_unit1_sent_3":
	{
		_this kbtell [_from, _topic,"conv_unit2_sent_4"];
	};

	case "conv_unit1_alter_yes":
	{
		_this kbtell [_from, _topic,"conv_unit2_confirm"];
	};

	case "conv_unit1_alter_no":
	{
		_this kbtell [_from, _topic,"conv_unit2_cancel"];
	};

	default {};
};

// Return the result to the scope
BIS_convMenu;
```

9. Create the file **conv_unit1.fsm** (for **unit1**):



FSM description for unit1:
NOTE: This **FSM** works when the **unit1** is controlled by AI.

Name	*Start*
Object type	Finite State
State type	**Start State**
InitCode	
PreCondition	

Name	*Reset*
Object type	Transition Condition
Condition type	**True Condition**
Priority	0
Condition	**true**
Action	
PreCondition	

Name	*End*
Object type	Finite State
State type	**End State**
InitCode	
PreCondition	

NOTE: Since FSM loops, it needs to have the *End* state for resetting the loop in idle through *Reset* condition.

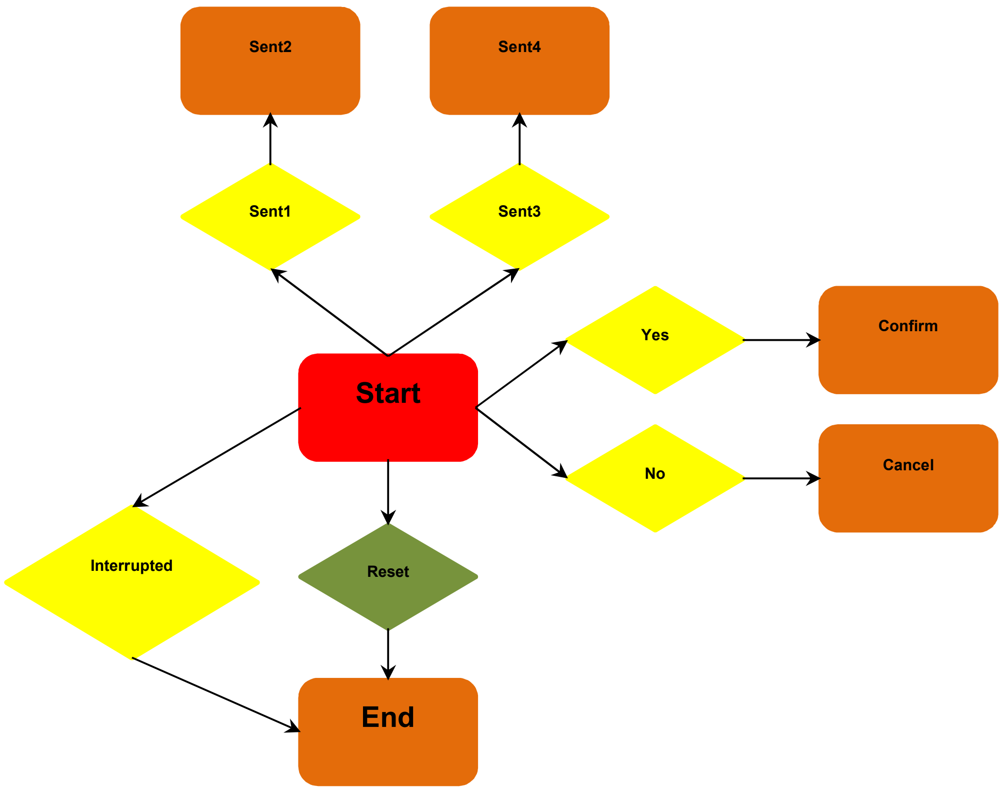
Name	*Sent2*
Object type	Transition Condition
Condition type	**Condition**
Priority	1
Condition	(_sentenceId in ["conv_unit2_sent_2"]);
Action	
PreCondition	

Name	*Interrupted*
Object type	Finite State
State type	**End State**
InitCode	_this kbTell [_from, _topic, "Interrupted"];
PreCondition	

Name	*Sent4*
Object type	Transition Condition
Condition type	**Condition**
Priority	1
Condition	(_sentenceId in ["conv_unit2_sent_4"]);
Action	
PreCondition	

Name	*No*
Object type	Finite State
State type	**End State**
InitCode	_this kbTell [_from, _topic, "conv_unit1_alter_no"];
PreCondition	

10. Create the file **conv_unit2.fsm** for **unit2**:



FSM description for unit2:

Name	*Start*
Object type	Finite State
State type	**Start State**
InitCode	
PreCondition	

Name	*Reset*
Object type	Transition Condition
Condition type	**True Condition**
Priority	0
Condition	**true**
Action	
PreCondition	

Name	*End*
Object type	Finite State
State type	**End State**
InitCode	
PreCondition	

NOTE: Since **FSM** loops, it needs to have the *End* state for resetting the loop in idle through *Reset* condition.

Name	*Interrupted*
Object type	Transition Condition
Condition type	**Condition**
Priority	1
Condition	(_sentenceId in ["Interrupted"]);
Action	
PreCondition	

Name	*Sent1*
Object type	Transition Condition
Condition type	**Condition**
Priority	1
Condition	(_sentenceId in ["conv_unit1_sent_1"]);
Action	
PreCondition	

Name	*Sent2*
Object type	Finite State
State type	**End State**
InitCode	_this kbTell [_from, _topic, "conv_unit2_sent_2"];
PreCondition	

Name	*Sent3*
Object type	Transition Condition
Condition type	**Condition**
Priority	1
Condition	(_sentenceId in ["conv_unit1_sent_3"]);
Action	
PreCondition	

Name	*Sent4*
Object type	Finite State
State type	**End State**
InitCode	_this kbTell [_from, _topic, "conv_unit2_sent_4"];
PreCondition	

Name	*Yes*
Object type	Transition Condition
Condition type	**Condition**
Priority	1
Condition	(_sentenceId in ["conv_unit1_alter_yes"]);
Action	
PreCondition	

Name	*Confirm*
Object type	Finite State
State type	**End State**
InitCode	_this kbTell [_from, _topic, "conv_unit2_confirm"];
PreCondition	

Name	*No*
Object type	Transition Condition
Condition type	**Condition**
Priority	1
Condition	(_sentenceId in ["conv_unit1_alter_no"]);
Action	
PreCondition	

Name	*Cancel*
Object type	Finite State
State type	**End State**
InitCode	_this kbTell [_from, _topic, "conv_unit2_cancel"];
PreCondition	

In the mission folder:

11. Create the file **init.sqf** (as instance).

In the file **init.sqf**

12. Add conversation topic to both the **unit1** and **unit2**.

```
unit1 kbAddTopic ["Dialog", "conv.bikb", "conv_unit1.fsm", {call compile preprocessFileLineNumbers "conv_unit1.sqf"}]
unit2 kbAddTopic ["Dialog", "conv.bikb", "conv_unit2.fsm", {call compile preprocessFileLineNumbers "conv_unit2.sqf"}]
```

NOTE:

- You can use any a script for this purpose.
- You can add a topic to any unit:

```
{if (side _x == Civilian) then {_x kbAddTopic [...]}} forEach allUnits
```

How to use

In mission editor:

1. Start the preview
2. Go to the character **unit2** so that the icon “Talk to” appears.
3. Scroll the mouse wheel.
4. In the quick action menu, choose “Say” item (“1” key).

You have to see the following sentences:

```
unit1: Hi! How are you?
unit2: Hi! All right!
```

5. The quick action menu appears.
6. In the quick action menu, choose “Continue” item (“1” key).

You have to see the following sentences:

```
unit1: You'll watch football today?
unit2: Yes! And you?
```

7. The quick action menu appears.
8. In the quick action menu, choose “Yes” item (“1” key).

You have to see the following sentences:

```
unit1: Yes, I'll watch football. Goodbye.
unit2: Good luck to your team. Goodbye.
```

Release: 2015.06.05
Publisher: http://vied-arma.ucoz.com/