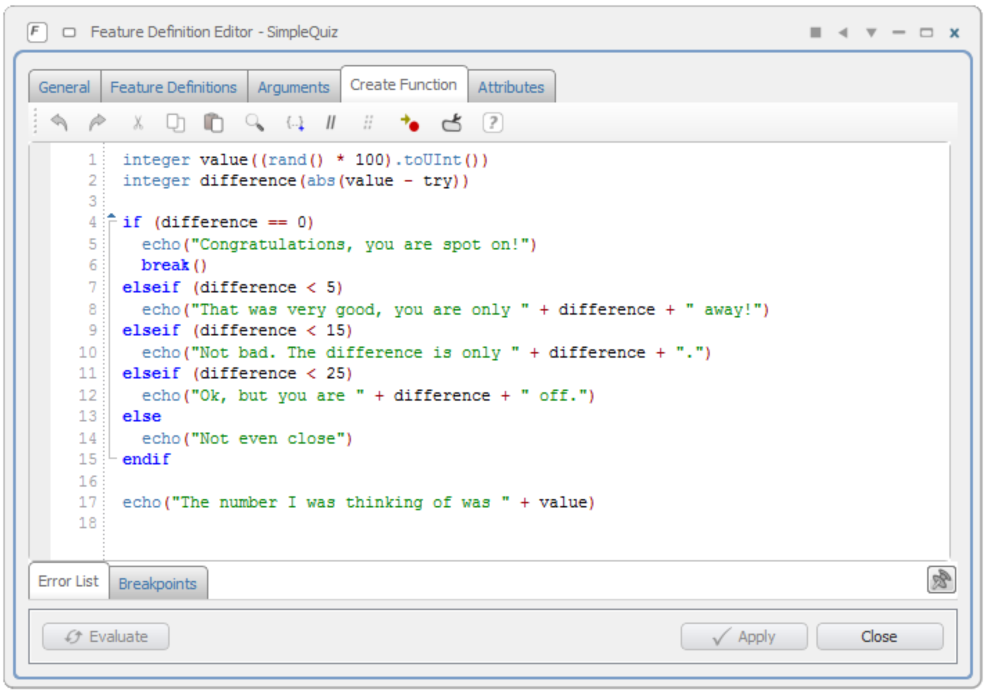
Case-by-Case Analysis

This tutorial introduces the feature programming language and shows how control structures can be used to perform case-by-case analysis.

As an example, a simple quiz is written. The player of the quiz (i.e. yourself) will try to match a number that gets generated by the feature.

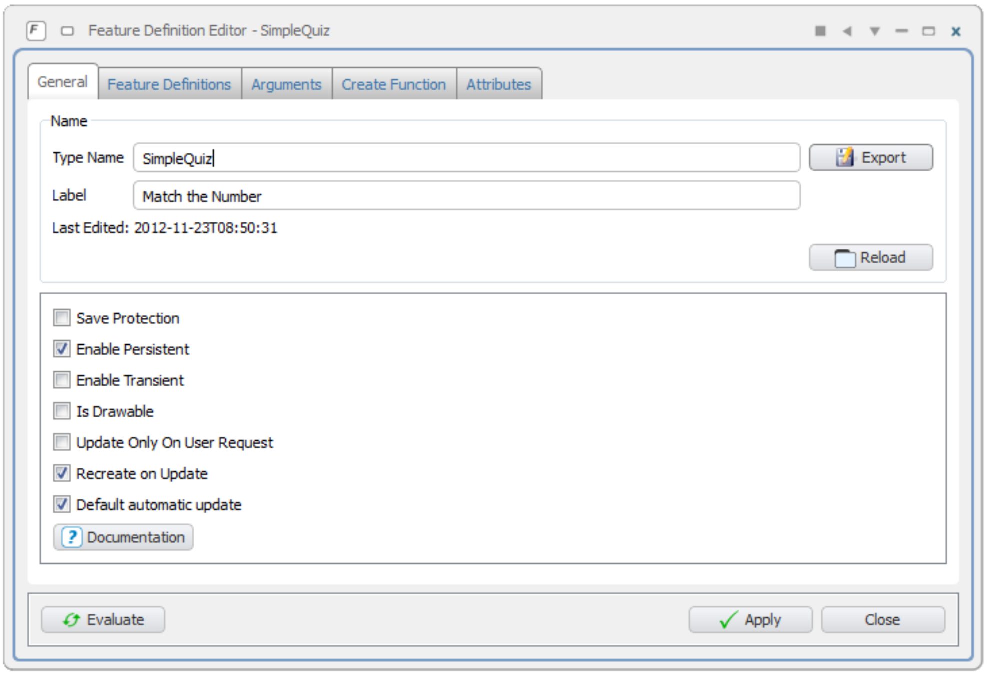


```
integer value((rand() * 100).toUInt())
integer difference(abs(value - try))

if (difference == 0)
  echo("Congratulations, you are spot on!")
  break()
elseif (difference < 5)
  echo("That was very good, you are only " + difference + " away!")
elseif (difference < 15)
  echo("Not bad. The difference is only " + difference + ".")
elseif (difference < 25)
  echo("Ok, but you are " + difference + " off.")
else
  echo("Not even close")
endif

echo("The number I was thinking of was " + value)
```

1 New Feature Definition

Remember: The *feature definition* is the "template" and represents the basis for the resulting *features.* It contains the (programmatic) description of the feature's behavior.

- Create a new definition by selecting *features > new definition.*
- Enter "SimpleQuiz" in the *type name* field (*general* tab).
- Enter "Match the Number" in the *label* field.
- Disable the *enable transient* checkbox.
- Disable the *is drawable* checkbox.
- Enable the *default automatic update* checkbox.


Feature Definition Editor - SimpleQuiz

Name
Type Name: SimpleQuiz
Label: Match the Number
Last Edited: 2012-11-23T08:50:31

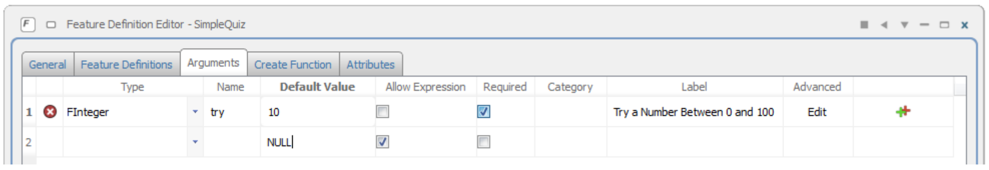
- [ ] Save Protection
- [x] Enable Persistent
- [ ] Enable Transient
- [ ] Is Drawable
- [ ] Update Only On User Request
- [x] Recreate on Update
- [x] Default automatic update

✓ This is a basic setup of the feature definition. It is called "SimpleQuiz" and will be labeled "Match the Number" in menus. It is a type definition since only persistent creation is enabled and it will not be displayed in the 3D-view because of disabling the checkbox *is drawable*.

Transient execution ("Enable Transient" in the screenshot) is equivalent to macro systems: it is a series of commands that get executed. No feature object is then available in the tree.

2 Arguments

First of all, our quiz needs a question. The input to the question is the argument that is then passed to the programmed logic.


Feature Definition Editor - SimpleQuiz

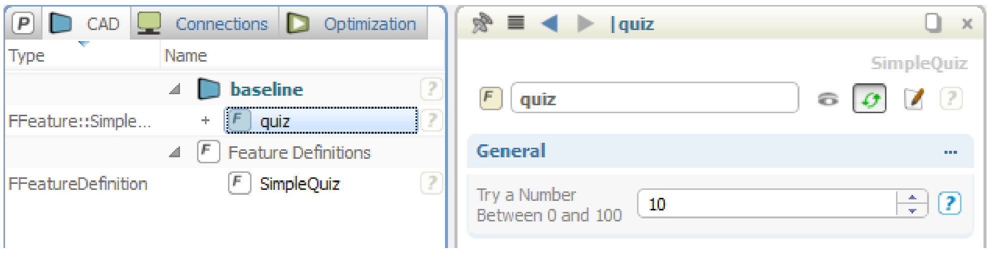
General | Feature Definitions | Arguments | Create Function | Attributes

	Type	Name	Default Value	Allow Expression	Required	Category	Label	Advanced
1	FInteger	try	10	☐	☑		Try a Number Between 0 and 100	Edit
2			NULL	☑	☐			

- ▶ Select the *arguments* tab.
- ▶ Select *FInteger* from the pull-down list in the column *type.*
- ▶ Enter “try” as name for the input.
- ▶ For the d*efault value,* enter “10”.
- ▶ Deselect *allow expression*.
- ▶ Select the *required* checkbox.
- ▶ Enter “Try a number between 0 and 100” into the column *label.*

Now we created the user interface for our little quiz. It’s time to take a first look at it.

- ▶ Press the *apply* button in the lower right corner of the dialog.
- ▶ Go to the object tree and open the node *feature definitions* from the *CAD* tree.
- ▶ Right click on the feature definition “SimpleQuiz” and select *create feature.*
- ▶ Open the node *baseline* and click on the newly created feature “f1”.
- ▶ Rename “f1” to “quiz”.



Type	Name
	baseline
FFeature::Simple…	quiz
	Feature Definitions
FFeatureDefinition	SimpleQuiz

Try a Number Between 0 and 100: 10

You can enter any integer number into the single input field of “quiz”. However, so far nothing happens when you do so. Now we need to implement the game logic.

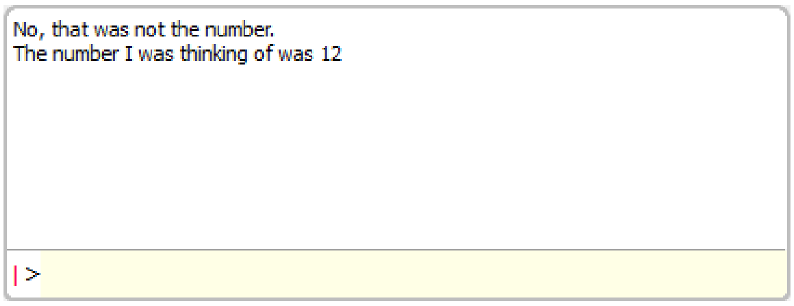
3 Create Function

Now that the question for our quiz is complete, we need to evaluate the "try" and give a result to the person taking the quiz. This is done in the command sequence defined in the *create function.*

- Go back to the *feature definition editor.*
- Select the *create function* tab.
- Paste the following text into the text editor – it will be explained line by line.

```
integer value((rand() * 100).toUInt())
integer difference(abs(value - try))

if (difference == 0)
  echo("Congratulations, you are spot on!")
else
  echo("No, that was not the number.")
  echo("The number I was thinking of was " + value)
endif
```

- Press *apply.*
- Go back to the object tree and select the feature "quiz" again. If it is already selected, deselect it and select it again (this refreshes the feature interface in the object editor).
- Enter a number between 0 and 100 in the text field and watch the *console* window.



```
No, that was not the number.
The number I was thinking of was 12
```

Depending on the quality of your try, the console output will either read "Congratulations, you are spot on!" or (more likely) "No that was not the number" and in a new line "The number I was thinking of was " followed by a number between 1 and 100. Now let us take a look at the code line by line.

4 Types and Conversion

Let's have a look at the first line. The number to match needs to be generated. This is done with this line:

```
integer value((rand() * 100).toUInt())
```

In order to understand that line, we need to analyze it starting from inside the most inner parentheses. The command `rand()` returns a randomly generated value between 0 and 1. Since we are looking for a number between 1 and 100, it is then multiplied by 100. As the number to match is supposed to be an integral value, and the result of the multiplication is still a floating point number, it is converted using the `toUInt()` command. The result of that command is then stored in a variable called "value" which is defined to be of type *unsigned integer*.

✓ Converting a floating point number to an integral value using the toInt() or toUInt() command cuts of the decimal places but performs no rounding. If you want to add rounding, you should add "0.5" to the floating point value before using the toInt() command. Alternativey, use the global commands *floor()* and *round()*. See the documentation browser for more information.

The integral type integer can also be abbreviated to *int*. Another integral type is *unsigned* (or *uint*) which represents positive integers only.

✓ As a shortcut to pressing the *apply* button you can also use the keyboard shortcut *CTRL+S*.

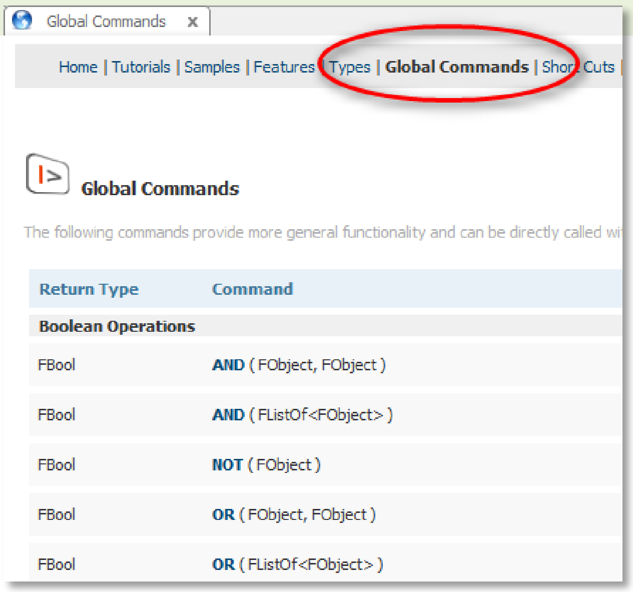
5 Types and Conversion

Now that the value to match has been generated, the next line calculates how far the user's "try" is off.

```
integer difference(abs(value - try))
```

Just like the first line, it is easiest to read the line from inside to outside. The difference between the try and the generated value is calculated by subtracting the two numbers from each other. Since we are only interested in how far the try is off, the `abs()` command is used which returns the absolute value of its argument. Finally, the difference is stored in a variable called "difference".

✓ There are more global commands like *abs()* available, see the documentation browser, *references > global commands.*


Global Commands
Home | Tutorials | Samples | Features | Types | Global Commands | Short Cuts |

Global Commands

The following commands provide more general functionality and can be directly called wi

Return Type	Command
Boolean Operations	
FBool	**AND** (FObject, FObject)
FBool	**AND** (FListOf<FObject>)
FBool	**NOT** (FObject)
FBool	**OR** (FObject, FObject)
FBool	**OR** (FListOf<FObject>)

6 Case-By-Case Analysis

This is where the actual case-by-case analysis starts. The feature "knows" how good the user's try was, but the user needs some feedback. This is done by using an *if/else* block. The basic syntax and functionality is as follows:

```
if (<condition>)
   // commands that are executed if the condition is true
else
   // commands that are executed if the condition is false
endif
```

Inside the parentheses following the `if` keyword there needs to be a logical statement that evaluates the boolean value as *true* or *false*. The commands between the line starting with `if` and the `else` keyword are executed if that logical statement is evaluated as *true*. If it is evaluated as *false*, the commands between the `else` keyword and the `endif` keyword are executed. Note that the `else` keyword is optional.

7 Operators and Echo

Let's have a look at the content of the different cases in the *create function*.

```
if (difference == 0)
```

The logical statement in this case compares the variable "difference" with the value "0" as that would mean that the user has matched the generated number. If this statement is true the line

```
echo("Congratulations, you are spot on!")
```

is executed which prints the success message to the console. However, if the difference is not equal to "0", the two lines

```
echo("No, that was not the number.")
echo("The number I was thinking of was " + value)
```

are executed telling the user that his try was wrong and what the actual value was. Note that the second line uses the "+" operator to concatenate the string "The number I was thinking of was" with the string representation of the integer variable "value".

✓ The example uses the equality operator "==" to compare two values.

Other possible comparison operators are "**!=**" **(**inequality), "<" (smaller than), "<=" (smaller or equal), ">" (larger) and ">=" (larger or equal than).

If you want to check multiple conditions at once, the following commands are available:

- "and(v1, v2)" is true when both, v1 and v2, evaluate as true
- "and([v1, v2, ..., vN])" is true when all values v1 to vN are true
- "or(v1, v2)" is true when either v1 or v2 evaluates as true
- "or([v1, v2, ..., vN])" is true when at least one of the values v1 to vN is true

8 More Rewards

Although the game is very simple, it is very hard to win, so in order to keep the user playing, we should add some more cases that reward the user with a positive message.

- ▶ Go back to the *create function* tab of the *feature definition editor*.
- ▶ Select all text in the editor and paste the following code into it.

```
integer value((rand() * 100).toUInt())
integer difference(abs(value - try))

if (difference == 0)
   echo("Congratulations, you are spot on!")
   break()
elseif (difference < 5)
   echo("That was very good, you are only " + difference + " away!")
elseif (difference < 15)
   echo("Not bad. The difference is only " + difference + ".")
elseif (difference < 25)
   echo("Ok, but you are " + difference + " off.")
else
   echo("Not even close")
endif

echo("The number I was thinking of was " + value)
```

- ▶ Press *apply* and play the game again.

Instead of having only two cases (won/lost) it now includes five cases. Note the usage of the `elseif` keyword. It behaves like a combination of `else` and `if`. If the previous condition does not apply, the condition of the `elseif` statement is evaluated and, if it is true, the commands following that line are executed until either `else`, `elseif` or `endif` is reached.

Another new statement is the `break()` command that was inserted into the if-block. It stops the execution at that point. It avoids having the final message (echo the original number to the console) since the user already knows the number.

9

Some Refactoring

Having a large number of if-cases can get quite confusing pretty fast. An alternative is using the *switch/case* syntax:

▶ Go back to the *create function* and replace the text with the following:

```
integer value((rand() * 100).toUInt())
integer difference(abs(value - try))

switch(difference)
case 0
   echo("Congratulations, you are spot on!")
   break()
case 1,2,3,4
   echo("That was very good, you are only " + difference + " away!")
case 5,6,7,8,9
   echo("Not bad. The difference is only " + difference + ".")
case 10,11,12,13,14
   echo("Ok, but you are " + difference + " off.")
default
   echo("Not even close")
endswitch

echo("The number I was thinking of was " + value)
```

▶ Press *apply* and test the game again (it should behave identically).

The value inside the parentheses behind the *switch* keyword is compared to the value(s) behind the *case* keyword(s). If they are equal the commands following that case line are executed. If no case applies the (optional) default branch is executed. If one case should apply to multiple values (like in our game) they can be separated by commas. Now it should be fairly easy to understand the new feature code.

The *switch/case* statement can be applied to all integral values (integer / unsigned integer) and string values. It may not only make it easier to understand the control flow of the code, but usually also results in faster execution compared to many nested *if/elseif* statements.