

# Visual driven database queries

What if you want to do powerful stuff and you are using HTML



**A visual database and flow management tool**

visual database

# Defining a model

The screenshot shows a Microsoft Dynamics 365 interface. At the top, there is a blue header bar with the text "ESUG 2023" and a dropdown arrow. Below the header, there is a tab labeled "todos" with a plus sign to its right. On the left side, there is a sidebar with icons for home, a grid, and a list. The main area displays a list view titled "Grid Ansicht" with a dropdown arrow. The list contains six entries with the following data:

	What to do?
1	one
2	two
3	three
4	four
5	five
6	six

Below the list, there is a summary row with a plus sign and the text "6 entries". To the right of the list, a column definition panel is open, showing a list of column types. The "Column Name" field is set to "state" with a character count of "5 / 25". The column types listed are:

- ABC Text
- 123 Number
- .00 Decimal
- Date and time
- Date
- Checkmark
- Single Select
- Multiple Select
- Link to a row of an other grid
- Links to multiple rows of an other grid
- Lookup

# Defining a model - adding an enum

The screenshot shows a Microsoft Dynamics 365 interface for a list named 'todos' under the 'ESUG 2023' entity. The list is displayed in 'Grid Ansicht' (Grid View) and has a column named 'What to do?' of type 'RBC'. The column contains six entries: 'one', 'two', 'three', 'four', 'five', and 'six'. A settings dialog is open for the 'What to do?' column, showing it is a 'Single Select' type. The dialog includes a 'Column Name' field with the value 'state', a 'Select Settings' section with three options: 'todo', 'in progress', and 'done', and a 'New Option' field. The 'Accept other options' toggle is turned off, and the 'Advanced settings' section is visible. The dialog has 'Cancel' and 'Save' buttons at the bottom.

	RBC What to do?
1	one
2	two
3	three
4	four
5	five
6	six
+ 6 entries	

Column Name: state (5 / 25)

Single Select

Select Settings

- todo
- in progress
- done

New Option

Accept other options

Advanced settings

Cancel Save

# Filtering & sorting

The screenshot shows a Microsoft Dynamics 365 interface. At the top, there is a blue header bar with "ESUG 2023" and a dropdown arrow. Below the header, there is a navigation pane on the left with a home icon, a "todos" dropdown menu, and a "Grid Ansicht" view selector. The main area displays a list view with a table of entries. A filter dialog is open over the table, showing a filter rule: "Where state does not equal done". The dialog also includes options to "Add filter" and "Add filter group".

ESUG 2023 ▾

todos ▾ +

Views

Grid Ansicht ▾

« Hide 1 Filter Sort Share Group ...

What to do

1	one	
2	two	
3	three	
4	four	todo
5	five	todo
6	six	todo

+ 6 entries

In this view, show entries

Where state does not equal done

+ Add filter + Add filter group

workflow engine

# Defining a flow - a trigger

The screenshot displays the 'New Flow' editor interface. At the top, a blue header bar contains a home icon, a back arrow, and the text 'New Flow' with an edit icon. Below the header, there are two tabs: 'Editor' (selected) and 'History'. The main workspace is a grid where a 'Trigger' icon is placed. The trigger icon is a blue circle with a white grid icon and a red exclamation mark, with the text 'Trigger' and 'A new Entry was added.' below it. On the right side, a configuration panel is open for the trigger. It has a close button (X), a grid icon, and the text 'A new Entry was added'. Below this, it says 'Trigger the flow when'. There are two dropdown menus: 'Choose Space' with 'ESUG 2023' selected, and 'Choose Event' with 'New Entry added' selected. Below these is a 'Choose Grid' section with a text input field containing 'todos'. At the bottom of the panel are 'Delete' and 'Save' buttons.



# Defining a flow - do an external call

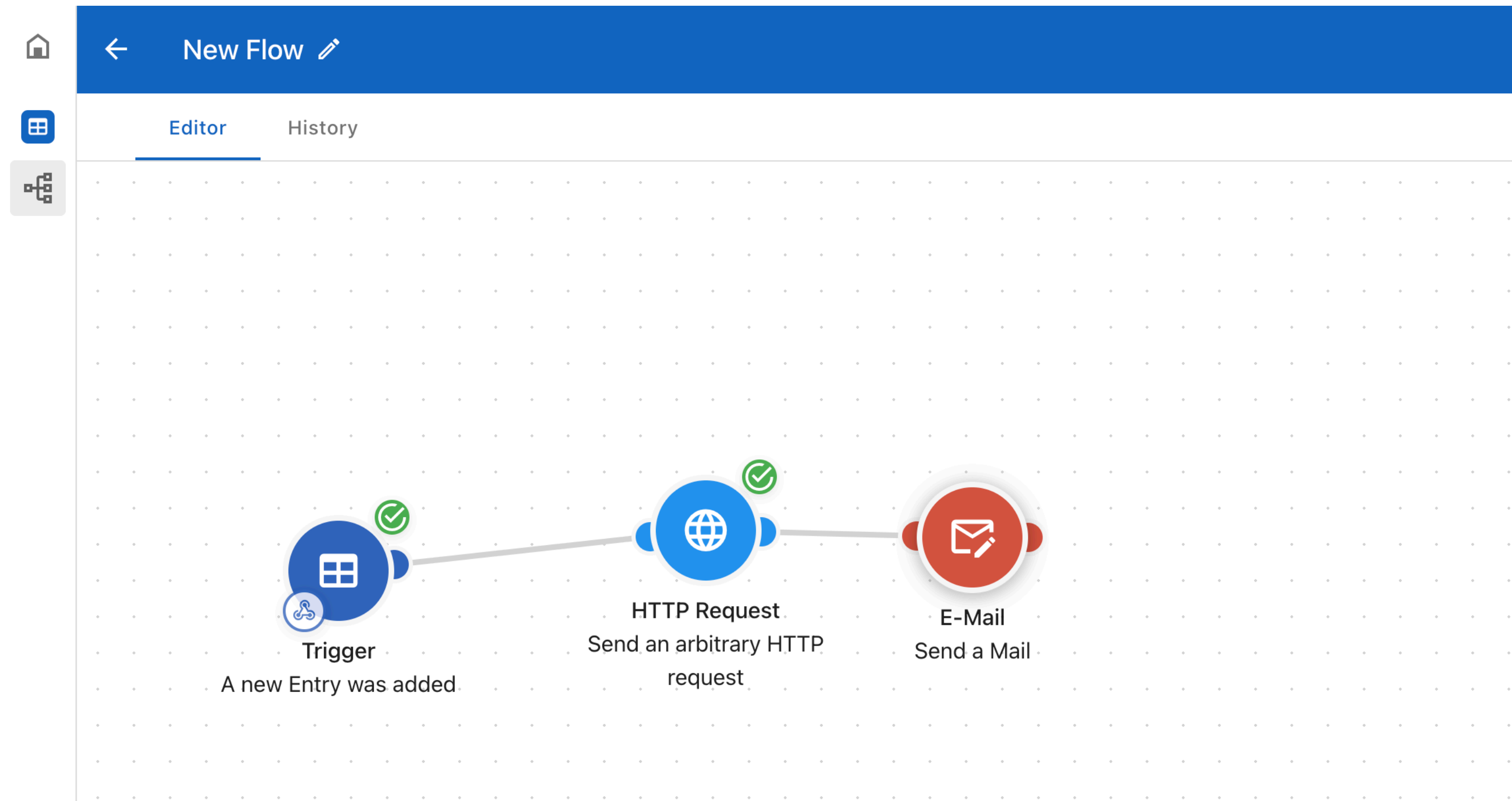
The screenshot shows the 'New Flow' editor interface. At the top, there is a blue header with a home icon, a back arrow, and the text 'New Flow' with an edit icon. Below the header, there are two tabs: 'Editor' (selected) and 'History'. The main workspace is a grid with a dotted pattern. A flow is defined with two steps connected by a line:

- Trigger:** A blue circle with a white grid icon. Below it, the text reads 'Trigger' and 'A new Entry was added.' A green checkmark is in the top right corner.
- HTTP Request:** A blue circle with a white globe icon. Below it, the text reads 'HTTP Request' and 'Send an arbitrary HTTP request.' A green checkmark is in the top right corner.

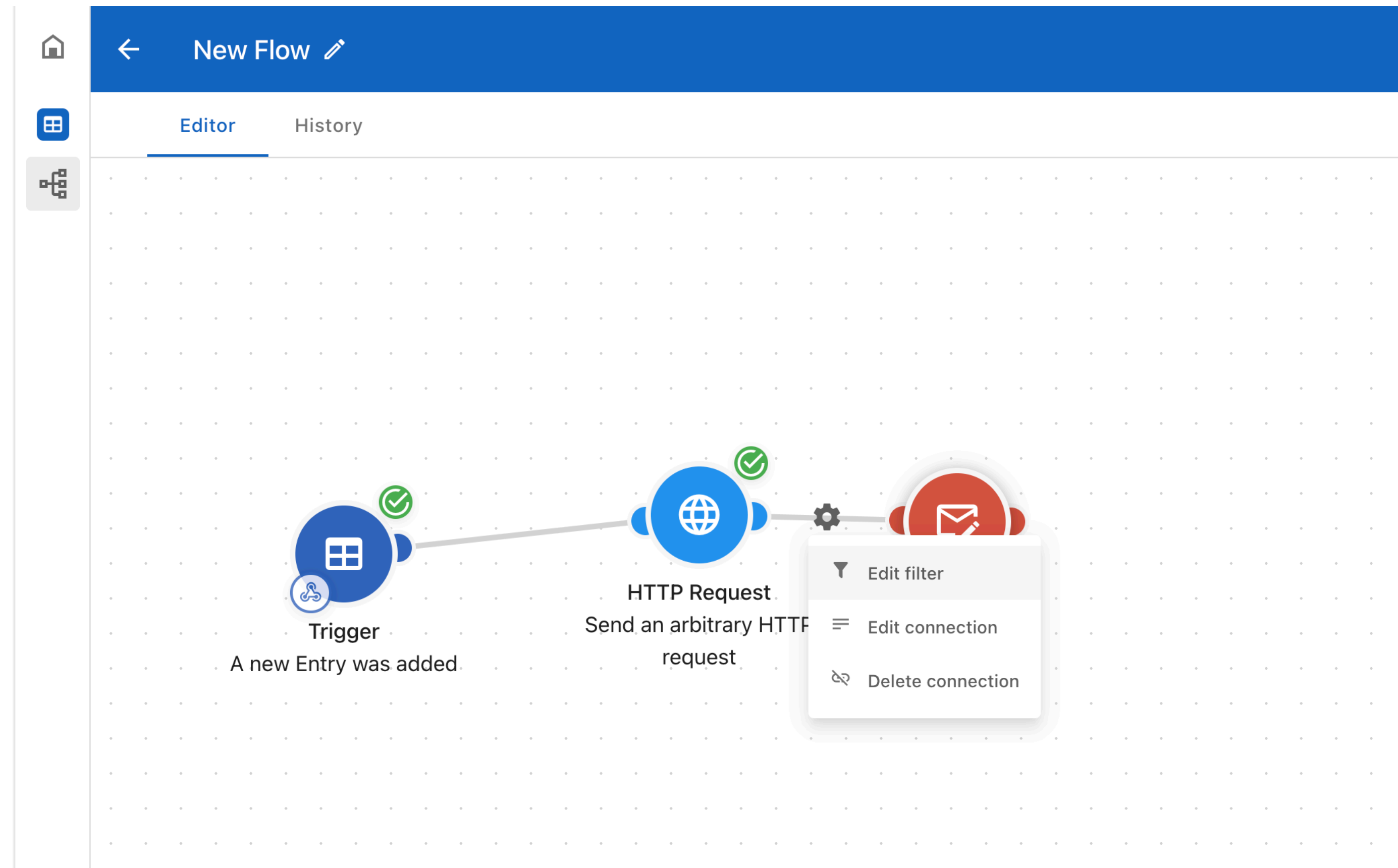
On the right side, there is a configuration panel for the 'Send an arbitrary HTTP request' step. It includes the following fields:

- URL:** A text input field containing 'http://norbert.hartl.name/esug.json'.
- Request Content:** A text input field containing 'Request Content'.
- Method:** A dropdown menu with 'GET' selected.
- Content Type:** A dropdown menu with 'Content Type' selected.

## Defining a flow - send a mail ...



## Defining a flow - ... with condition...



## Defining a flow - visually pick whatever you got

The screenshot shows a flow editor interface with a dark blue header and a grey grid workspace. The header contains a home icon, a back arrow, and the text 'New Flow' with an edit icon. Below the header, there are tabs for 'Editor' and 'History'. On the left side, there is a sidebar with icons for a grid, a flow, and a refresh. In the workspace, a blue circular 'Trigger' node is visible, with a green checkmark and a plus sign. Below it, the text 'Trigger' and 'A new Entry was added.' is displayed. Two dialog boxes are open over the workspace. The first dialog, titled 'Only continue if...', has a blue header and contains a text input field with 'state', a dropdown menu labeled 'Choose condition...', a text input field labeled 'Enter text ...', and a checkbox labeled 'Show filter expression'. A 'Cancel' button is at the bottom right. The second dialog, titled 'Available fields', has a blue header and a close button. It lists various fields under different categories: 'A new Entry was added:', 'Send an arbitrary HTTP request:', 'output', 'data', 'state done', 'headers', 'Accept-Ranges bytes', 'Last-Modified Tue, 29 Aug 2023 15:29:28 G...', 'Etag "64ee0ed8-16"', 'Content-Length 22', 'Connection keep-alive', 'Server nginx/1.18.0 (Ubuntu)', 'Date Tue, 29 Aug 2023 15:31:14 GMT', 'Content-Type application/json', and 'code 200'.

## Defining a flow - define condition

The screenshot displays a software interface for defining a flow. At the top, a dark blue header contains a home icon, a back arrow, and the text "New Flow" with an edit icon. Below the header, there are two tabs: "Editor" (selected) and "History". The main workspace is a grey grid. On the left, a "Trigger" node is shown with a blue circular icon containing a grid and a checkmark. Below it, the text "Trigger" and "A new Entry was added." is visible. A dialog box titled "Only continue if..." is open in the center. The dialog has a blue header and contains the following fields: a text input with "state", a dropdown menu with "is equal to" and a close button (X), a text input with "done", and a dropdown menu with "Show filter expression" and a downward arrow. At the bottom of the dialog are "Cancel" and "Save" buttons.

# Uniform approach

- has knowledge about column types and operations
- has syntax for composing type based filters
- turns JSON into a tree that generates access path
- knows about application scope

# ApptiveScript

- has knowledge about column types and operations
- has syntax for composing type based filters
- turns JSON into a tree that generates access path
- knows about application scope
- is an executable DSL for ApptiveGrid

## Defining a flow - visually pick whatever you got

The screenshot shows a flow editor interface with a dark blue header and a grey grid background. The header contains a home icon, a back arrow, and the text 'New Flow' with an edit icon. Below the header, there are two tabs: 'Editor' (active) and 'History'. On the left side, there is a sidebar with a grid icon and a plus icon. In the center of the grid, there is a blue circular node labeled 'Trigger' with a green checkmark and a plus icon. Below the node, the text 'Trigger' and 'A new Entry was added.' is visible. Two dialog boxes are open over the grid. The first dialog box is titled 'Only continue if...' and has a blue header. It contains a text input field with 'state' entered, a dropdown menu labeled 'Choose condition...', a text input field labeled 'Enter text ...', and a checkbox labeled 'Show filter expression'. A 'Cancel' button is at the bottom right. The second dialog box is titled 'Available fields' and has a blue header with a close icon. It contains a list of fields with their values: 'A new Entry was added:', 'Send an arbitrary HTTP request:', 'output', 'data', 'state done', 'headers', 'Accept-Ranges bytes', 'Last-Modified Tue, 29 Aug 2023 15:29:28 G...', 'Etag "64ee0ed8-16"', 'Content-Length 22', 'Connection keep-alive', 'Server nginx/1.18.0 (Ubuntu)', 'Date Tue, 29 Aug 2023 15:31:14 GMT', 'Content-Type application/json', and 'code 200'.



## Combined syntax

Only continue if...

state

is equal to

×

done

Show filter expression ^

```
step('64ee0ddae4743dbe24c06aee').output.get('data').get('state').isEqualTo('done')
```

Cancel

Save

`step('64ee0ddae4743dbe24c06aee').output.get('data').get('state').isEqualTo('done')`

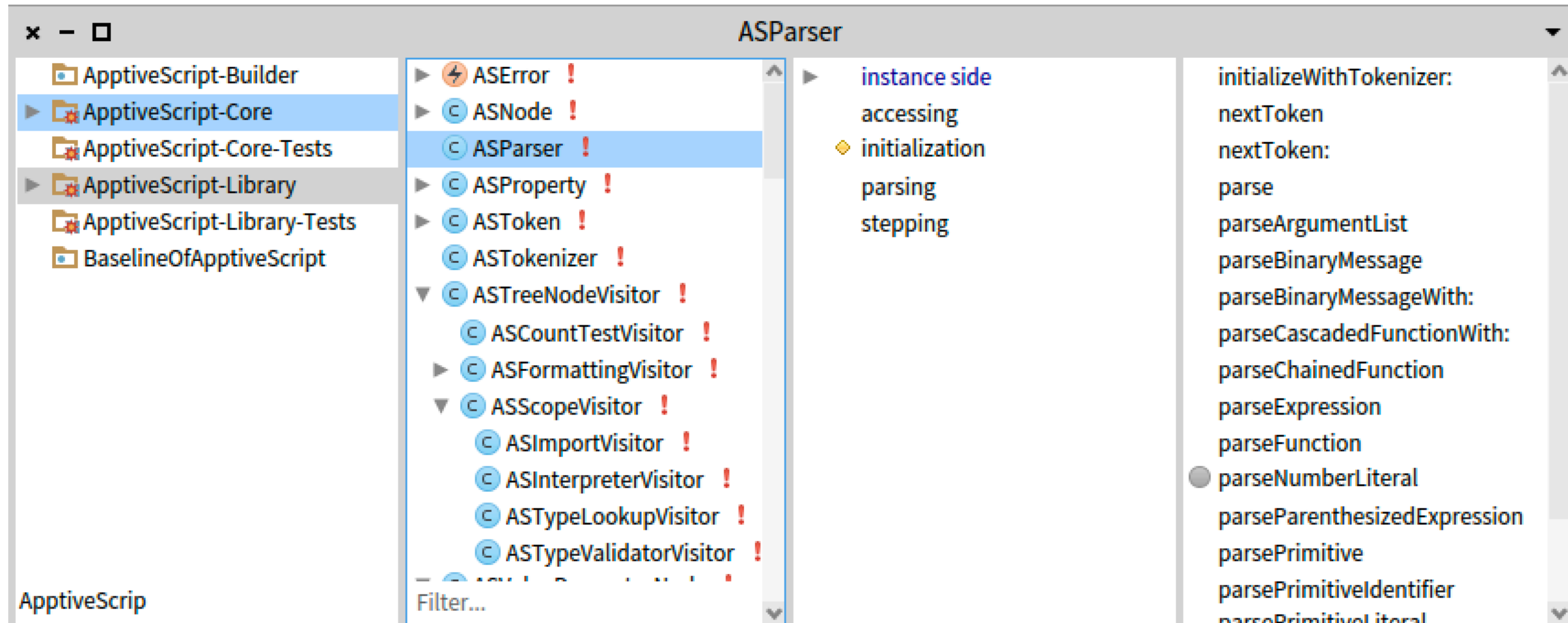
application

node

user

type

# ApptiveScript



# ApptiveScript

The screenshot shows the ApptiveScript Playground interface. The main editor contains three lines of code:

```
1 ApptiveScript parse:  
  'step('64ee0ddae4743dbe24c06aee').output.get('data')  
  .get('state').isEqualTo('done')'.  
2  
3 ApptiveScriptBuilder build .
```

The right-hand side of the interface features a variable inspector with tabs for 'Raw', 'Breakpoints', and 'Meta'. The 'Raw' tab is active, displaying a tree view of variables and their values:

Variable	Value
self	step('64ee0ddae4743dbe24c06aee').output.get('data')
{ } properties	a Dictionary [0 items] ()
function	step('64ee0ddae4743dbe24c06aee').output.get('data')
self	step('64ee0ddae4743dbe24c06aee').output.get('data')
{ } properties	a Dictionary [0 items] ()
receiver	step('64ee0ddae4743dbe24c06aee').output.get('data')
self	step('64ee0ddae4743dbe24c06aee').output.get('data')
{ } properties	a Dictionary [0 items] ()
function	step('64ee0ddae4743dbe24c06aee').output.get('data')
self	step('64ee0ddae4743dbe24c06aee').output.get('data')
{ } properties	a Dictionary [0 items] ()
receiver	step('64ee0ddae4743dbe24c06aee').output.get('data')
property	get
arguments	an OrderedCollection [1 item] ('state')
property	isEqualTo

Below the variable inspector, a small code editor shows the current line of code being inspected: `1 self`. The status bar at the bottom left indicates 'Line: 1:1'.

# ApptiveScript - *Phar*JS

The screenshot shows an IDE window titled "ApptiveScriptBuilder>>build:". The left sidebar displays a project tree with folders: ApptiveScript-BUILDER, ApptiveScript-Core, ApptiveScript-Core-Tests, ApptiveScript-Library, ApptiveScript-Library-Test, and BaselineOfApptiveScript. The main area is divided into three panes: a class hierarchy pane showing AGFileExporter, ASTRanspiler, and ApptiveScriptBuilder; a class side pane showing instance side, building, and compiling; and a methods pane showing build, build:, compilePublicPropertySelecto, and compilePublicPropertySelecto. Below the panes are navigation options: All Packages, Scoped View, Flat, Hier., Inst. side, Class side, Methods, Vars, Class refs., and Impleme. The editor shows the following code for the build method:

```
build: aString
  (AGFileExporter newWithAppClass: ApptiveScript)
  path: aString asFileReference;
  transpiler: (ASTranspiler new pharoJsSelectorPrefix: 'as_');
  writeFiles
```

At the bottom, there is a status bar with "1/6 [1]" and "building extension F +L W". A utility methods section is also visible at the bottom left.

# ApptiveScript on the frontend

- generated picking visual elements
- enables auto completion through type information.
- can be executed on demand
- sends syntax to the server

# ApptiveScript on the backend

- is parsed to AST
- AST is stored in soil
- uses the AST as database predicate

# ApptiveScript on the backend

- objects are read using an iterator (#next just as in stream)
- iterator executes the ApptiveScript AST on object
- uses the AST as predicate
- returns the result



Thank you!  
Questions?