Francy

Framework for Interactive Discrete Mathematics

1.2.4

17 May 2019

Manuel Martins

Manuel Martins
Email: manuelmachadomartins@gmail.com
Homepage: http://github.com/mcmartins
Address: Departamento de Ciências e Tecnologia da Universidade
Aberta
Lisboa, Portugal
Faculdade de Ciências e Tecnologia da Universidade de
Coimbra
Coimbra, Portugal

Contents

Chapter 1

Introduction

1.1 Francy

Francy arose from the necessity of having a lightweight framework for building interactive graphics, generated from GAP, running primarily on the web, primarily in a Jupyter Notebook. An initial attempt to re-use XGAP and port it was made, but the lack of a standardized data exchange format between GAP and the graphics renderer, and the simplistic initial requirements of the project were the basis for the creation of a new GAP package. Take a look at Francy functionality on these Jupyter Notebooks.

1.2 Applications

Francy has potentially many applications and can be used to provide a graphical representation of data structures, allowing one to navigate through and explore properties or relations of these structures. In this way, Francy can be used to enrich a learning environment where GAP provides a library of thousands of functions implementing algebraic algorithms as well as large data libraries of algebraic objects. FrancyMonoids and SubgroupLattice are some example packages using Francy.

1.3 Functionality

Francy provides an interface to draw graphics using objects. This interface is based on simple concepts of drawing and graph theory, allowing the creation of directed and undirected graphs, trees, line charts, bar charts and scatter charts. These graphical objects are drawn inside a canvas that includes a space for menus and to display informative messages. Within the canvas it is possible to interact with the graphical objects by clicking, selecting, dragging and zooming.

1.4 Installation

This package requires the GAP packages JupyterKernel and json, all of which are distributed with GAP. Francy follows a similar installation procedure to JupyterKernel, so it requires Jupyter to be installed on your system. Please note, you need to run the installation commands from the same python version Jupyter is installed on. In order to install/update Francy, please run the following command to download the latest version available from `https://pypi.org/`:

Example

```
        mcmartins@local:~$ pip install jupyter_francy -U
```

It is necessary to enable Francy on your Jupyter Notebook installation:

Example

```
        mcmartins@local:~$ jupyter nbextension enable --py --sys-prefix jupyter_francy
```

For Jupyter Lab, please run:

Example

```
        mcmartins@local:~$ jupyter labextension build
```

Alternatively, if you want to run Francy only on Jupyter Lab, simply execute:

Example

```
        mcmartins@local:~$ jupyter labextension install jupyter-francy
```

This will load the module from `https://npmjs.org`. This approach should be used if you want to run Francy only on Jupyter Lab.

1.5 How it works

Francy requires a rendering package to display graphics. Francy uses Renderers based on D3.js and Graphviz, for rendering the semantic representation produced by the GAP package. The renderers can be swithed at any time using the user interface, by selecting 'Settings > Renderers' in the main menu. This library is distributed both as a browser module and as a Jupyter extension. The Jupyter extension can be used in Jupyter Notebook or Jupyter Lab, using the JupyterKernel and the MIME type 'application/vnd.francy+json' to render the document. Please check the JavaScript Documentation for more information.

1.6 Publications

ICMS 2018

Chapter 2

Francy Callbacks

`Callbacks` are objects that hold a function, a list of arguments and a trigger event. `Callbacks` are used to execute GAP code from a remote client using the `Trigger` Operation.

`Callbacks` can be added directly to `Menus` and/or `Shapes`.

Please see Francy-JS for client implementation.

2.1 Categories

In this section we show all Francy Callback Categories.

2.1.1 IsCallback (for IsFrancyObject)

▷ `IsCallback(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `Callback` objects.

2.1.2 IsRequiredArg (for IsFrancyObject)

▷ `IsRequiredArg(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `RequiredArg` objects.

2.1.3 IsArgType (for IsFrancyTypeObject)

▷ `IsArgType(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `ArgType` objects.

2.1.4 IsTriggerType (for IsFrancyTypeObject)

▷ `IsTriggerType(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `TriggerType` objects.

2.2 Families

In this section we show all Francy Callback Families.

2.3 Representations

In this section we show all Francy Callback Representations.

2.3.1 IsCallbackRep (for IsComponentObjectRep)

▷ `IsCallbackRep(`*`arg`*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `Callback` internal representation.

2.3.2 IsRequiredArgRep (for IsComponentObjectRep)

▷ `IsRequiredArgRep(`*`arg`*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `RequiredArg` internal representation.

2.3.3 IsArgTypeRep (for IsComponentObjectRep)

▷ `IsArgTypeRep(`*`arg`*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `ArgType` internal representation.

2.3.4 IsTriggerTypeRep (for IsComponentObjectRep)

▷ `IsTriggerTypeRep(`*`arg`*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `TriggerType` internal representation.

2.4 Operations

In this section we show all Francy Callback Operations.

2.4.1 Callback (for IsTriggerType, IsFunction, IsList)

▷ `Callback(`*`IsTriggerType, IsFunction, IsList(object)`*`)` (operation)

Returns: `Callback`

Creates a Callback object that holds a function and args to be executed by a trigger based on a trigger type.

Please note, the Function must Return!

Examples:

Create a simple `Callback` with no arguments:

Example

```
gap> MyFunction := function() return "Hello World!"; end;
gap> callback := Callback(MyFunction);
gap> Id(callback);
```

Create a `Callback` with one required argument of type string:

Example

```
gap> MyFunction := function(str) return Concatenation("Hello", " ", str); end;
gap> callback := Callback(MyFunction);
gap> arg := RequiredArg(ArgType.STRING, "Your Name");
gap> Add(callback, arg);
```

Create a `Callback` with one known argument of type string:

Example

```
gap> MyFunction := function(args) return args; end;
gap> callback := Callback(MyFunction, ["Hello World"]);
```

Create a `Callback` with one known argument and one required argument of type string:

Example

```
gap> MyFunction := function(a,b) return Concatenation(a, b); end;
gap> callback := Callback(MyFunction, ["Hello "]);
gap> arg := RequiredArg(ArgType.STRING, "Your Name");
gap> Add(callback, arg);
```

Create a `Callback` with one known argument and one required argument of type string and double click trigger Type:

Example

```
gap> MyFunction := function(a,b) return Concatenation(a, b); end;
gap> callback := Callback(TriggerType.DOUBLE_CLICK, MyFunction, ["Hello "]);
gap> arg := RequiredArg(ArgType.STRING, "Your Name");
gap> Add(callback, arg);
```

In order to see the output of the previous examples, we have to simulate the external call to `Trigger` operation:

Example

```
gap> MyFunction := function(a,b) return Concatenation(a, b); end;
gap> callback := Callback(TriggerType.DOUBLE_CLICK, MyFunction, ["Hello "]);
gap> arg := RequiredArg(ArgType.STRING, "Your Name");
gap> SetTitle(arg, "Enter your name");
gap> Title(arg);
gap> Add(callback, arg);
gap> SetValue(arg, "Manuel"); # simulate the user input
gap> Value(arg);
gap> Trigger(GapToJsonString(Sanitize(callback))); # simulate the external trigger
```

Create a Noop Callback, useful for Menu holders, where no function is required:

Example

```
gap> callback := NoopCallback();
```

2.4.2 NoopCallback

▷ `NoopCallback()` (operation)

Returns: `Callback`

Creates an empty Callback object that does nothing. Useful to create menu holders.

2.4.3 RequiredArg (for IsArgType, IsString)

▷ `RequiredArg(`*`IsArgType, IsString(title)`*`)` (operation)

Returns: `RequiredArg`

Creates a `Callback` `RequiredArg`. `RequiredArg` is user input driven and required for the execution of a `Callback` This value will be provided by the user.

2.4.4 Trigger (for IsString)

▷ `Trigger(`*`IsString(JSON)`*`)` (operation)

Returns: the result of the execution of the `Callback` defined `Function`

Triggers a callback function in GAP. Gets a JSON String object representation of the callback to execute.

2.4.5 Add (for IsCallback, IsRequiredArg)

▷ `Add(`*`IsCallback[, IsRequiredArg, List(IsRequiredArg)]`*`)` (operation)

Returns: `Callback`

Adds a `RequiredArg` to a specific `Callback`.

2.4.6 Remove (for IsCallback, IsRequiredArg)

▷ `Remove(`*`IsCallback[, IsRequiredArg, List(IsRequiredArg)]`*`)` (operation)

Returns: `Callback`

Removes a `RequiredArg` from a specific callback.

2.5 Globals

In this section we show the Global Callback Francy Records for multi purpose.

2.6 Attributes

In this section we show the Francy Callback Attributes

2.6.1 Title (for IsRequiredArg)

▷ `Title(`*`arg`*`)` (attribute)

Returns: `IsString` with the title of the object

A title on a required arg is used to ask the user what is expected from his input.

2.6.2 Title (for IsRequiredArg)

▷ `Title(`*`arg1`*`)` (function)

2.6.3 SetTitle (for IsRequiredArg, IsString)

▷ `SetTitle(`*`IsRequiredArg, IsString`*`)` (function)

Sets the title of the required arg.

2.6.4 Value (for IsRequiredArg)

▷ `Value(`*`arg`*`)` (attribute)

Returns: `IsString` with the value of the object

A value on a required arg is the actual input to be passed to gap. These values are stored as `String` for convenience, even if the `ArgType` specified for the `RequiredArg` is another. Explicit conversion is required within the `Callback`function.

2.6.5 Value (for IsRequiredArg)

▷ `Value(`*`arg1`*`)` (function)

2.6.6 SetValue (for IsRequiredArg, IsString)

▷ `SetValue(`*`IsRequiredArg, IsString`*`)` (function)

Sets the value of the required arg.

2.6.7 ConfirmMessage (for IsCallback)

▷ `ConfirmMessage(`*`arg`*`)` (attribute)

Returns: `IsString` with the message oto be shown to the user prior to the callback execution

This will display a confirmation message before any callback is executed.

2.6.8 ConfirmMessage (for IsCallback)

▷ `ConfirmMessage(`*`arg1`*`)` (function)

2.6.9 SetConfirmMessage (for IsCallback, IsString)

▷ `SetConfirmMessage(`*`IsRequiredArg, IsString`*`)` (function)

Sets the value of the message to display to the user.

Chapter 3

Francy Canvas

A `Canvas` is an area where the graphics representation of Francy live.

Please see Francy-JS for client implementation.

3.1 Categories

In this section we show all Francy Canvas Categories.

3.1.1 IsCanvas (for IsFrancyObject)

▷ `IsCanvas(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `Canvas` objects.

3.1.2 IsCanvasDefaults (for IsFrancyDefaultObject)

▷ `IsCanvasDefaults(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `CanvasDefaults` objects.

3.2 Families

In this section we show all Francy Canvas Families.

3.3 Representations

In this section we show all Francy Canvas Representations.

3.3.1 IsCanvasRep (for IsComponentObjectRep)

▷ `IsCanvasRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `Canvas` internal representation.

3.3.2 IsCanvasDefaultsRep (for IsComponentObjectRep)

▷ `IsCanvasDefaultsRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `CanvasDefaults` internal representation.

3.4 Operations

In this section we show all Francy Canvas Operations.

3.4.1 Canvas (for IsString, IsCanvasDefaults)

▷ `Canvas(`*IsString(title)[, IsCanvasDefaults]*`)` (operation)

Returns: `Callback`

Canvas represents a base element to draw graphics on. Inspired by the HTML canvas tag element which is used to draw graphics, in runtime, via JavaScript. Examples:

Create a simple `Canvas`:

Example

```
gap> canvas := Canvas("");
gap> Id(canvas);
gap> SetTitle(canvas, "Quaternion Group Subgroup Lattice");
gap> Title(canvas);
gap> SetHeight(canvas, 400); # default 600
gap> Height(canvas);
gap> SetWidth(canvas, 400); # default 800
gap> Width(canvas);
gap> SetZoomToFit(canvas, false); # default true
gap> ZoomToFit(canvas);
gap> Draw(canvas);
```

3.4.2 Add (for IsCanvas, IsFrancyGraph)

▷ `Add(`*IsCanvas[, IsFrancyGraph, List(IsFrancyGraph)]*`)` (operation)

Returns: `Canvas`

Adds a `FrancyGraph` to a specific `Canvas`. Well, the api is abstract enough to allow Adding a list of `IsFrancyGraph` to a canvas, but this results in setting the graph property only to the last `IsFrancyGraph` in the list.

3.4.3 Remove (for IsCanvas, IsFrancyGraph)

▷ `Remove(`*IsCanvas[, IsFrancyGraph, List(IsFrancyGraph)]*`)` (operation)

Returns: `Canvas`

Removes a `FrancyGraph` from a `Canvas`.

3.4.4 Add (for IsCanvas, IsChart)

▷ `Add(`*IsCanvas[, IsChart, List(IsChart)]*`)` (operation)

Returns: `Canvas`

Adds a `Chart` to a specific `Canvas`. Well, the api is abstract enough to allow Adding a list of `IsChart` to a canvas, but this results in setting the graph property only to the last `IsChart` in the list.

3.4.5 Remove (for IsCanvas, IsChart)

▷ `Remove(`*IsCanvas[, IsChart, List(IsChart)]*`)` (operation)

Returns: `Canvas`

Removes a `Chart` from a `Canvas`.

3.4.6 Add (for IsCanvas, IsMenu)

▷ `Add(`*IsCanvas[, IsMenu, List(IsMenu)]*`)` (operation)

Returns: `Canvas`

Adds a `Menu` to a specific `Canvas`.

3.4.7 Remove (for IsCanvas, IsMenu)

▷ `Remove(`*IsCanvas[, IsMenu, List(IsMenu)]*`)` (operation)

Returns: `Canvas`

Removes a `Menu` from a `Canvas`.

3.4.8 Add (for IsCanvas, IsFrancyMessage)

▷ `Add(`*IsCanvas[, IsFrancyMessage, List(IsFrancyMessage)]*`)` (operation)

Returns: `IsCanvas`

Adds a `FrancyMessage` to a specific `IsCanvas`.

3.4.9 Remove (for IsCanvas, IsFrancyMessage)

▷ `Remove(`*IsCanvas[, IsFrancyMessage, List(IsFrancyMessage)]*`)` (operation)

Returns: `IsCanvas`

Removes a `FrancyMessage` from a specific `IsCanvas`.

3.4.10 Draw (for IsCanvas)

▷ `Draw(`*IsCanvas*`)` (operation)

Returns: `rec` with json representation of the canvas

Generates the JSON representation of the canvas and children objects

3.4.11 DrawSplash (for IsCanvas)

▷ `DrawSplash(`*IsCanvas*`)` (operation)

Returns: `rec` with html generated

Generates an HTML page and opens it within the default browser of the system

3.5 Global

In this section we show all Global Francy Canvas Records for multi purpose.

3.6 Attributes

In this section we show the Francy Attributes

3.6.1 Width (for IsCanvas)

▷ `Width(`*arg*`)` (attribute)

Returns: `IsPosInt`

The Width of the canvas in pixels

3.6.2 Width (for IsCanvas)

▷ `Width(`*arg1*`)` (function)

3.6.3 SetWidth (for IsCanvas, IsPosInt)

▷ `SetWidth(`*IsCanvas, IsPosInt*`)` (function)

Sets the Width of the canvas in pixels

3.6.4 Height (for IsCanvas)

▷ `Height(`*arg*`)` (attribute)

Returns: `IsPosInt`

The Height of the canvas in pixels

3.6.5 Height (for IsCanvas)

▷ `Height(`*arg1*`)` (function)

3.6.6 SetHeight (for IsCanvas, IsPosInt)

▷ `SetHeight(`*IsCanvas, IsPosInt*`)` (function)

Sets the Height of the canvas in pixels

3.6.7 ZoomToFit (for IsCanvas)

▷ `ZoomToFit(`*arg*`)` (attribute)

Returns: `IsBool` True if enabled otherwise False

`ZoomToFit` is a property that sets the zoom to fit behavior on change in the client implementation.

3.6.8 ZoomToFit (for IsCanvas)

▷ `ZoomToFit(`*arg1*`)` (function)

3.6.9 SetZoomToFit (for IsCanvas, IsBool)

▷ `SetZoomToFit(`*`IsCanvas, IsBool`*`)` (function)

`ZoomToFit` is a property that sets the zoom to fit behavior on change in the client.

3.6.10 Title (for IsCanvas)

▷ `Title(`*`arg`*`)` (attribute)

Returns: `IsString` with the title of the object
A title on a required arg is used to ask the user what is expected from his input.

3.6.11 Title (for IsCanvas)

▷ `Title(`*`arg1`*`)` (function)

3.6.12 SetTitle (for IsCanvas, IsString)

▷ `SetTitle(`*`IsCanvas, IsString`*`)` (function)

Sets the title of the required arg.

3.6.13 TexTypesetting (for IsCanvas)

▷ `TexTypesetting(`*`arg`*`)` (attribute)

Returns: `IsBool` with the title of the object
Enables usage of Tex typestting on the client implementation, if supported.

3.6.14 TexTypesetting (for IsCanvas)

▷ `TexTypesetting(`*`arg1`*`)` (function)

3.6.15 SetTexTypesetting (for IsCanvas, IsBool)

▷ `SetTexTypesetting(`*`IsCanvas, IsBool`*`)` (function)

Sets Tex typestting on the canvas objects

Chapter 4

Francy Charts

It is possible to build `Charts` with simple `Datasets`.

Currently, Francy, supports Bar, Line and Scatter Charts.

Please see Francy-JS for client implementation.

4.1 Categories

In this section we show all Francy Chart Categories.

4.1.1 IsChart (for IsFrancyObject)

▷ `IsChart(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `Chart` objects.

4.1.2 IsChartType (for IsFrancyTypeObject)

▷ `IsChartType(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `ChartType` objects.

4.1.3 IsChartDefaults (for IsFrancyDefaultObject)

▷ `IsChartDefaults(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `ChartDefaults` objects.

4.1.4 IsAxisScaleType (for IsFrancyTypeObject)

▷ `IsAxisScaleType(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `AxisScaleType` objects.

4.1.5 IsXAxis (for IsFrancyObject)

▷ `IsXAxis(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `XAxis` objects.

4.1.6 IsYAxis (for IsFrancyObject)

▷ `IsYAxis(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `YAxis` objects.

4.1.7 IsDataset (for IsFrancyObject)

▷ `IsDataset(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `Dataset` objects.

4.2 Families

In this section we show all Francy Chart Families.

4.3 Representations

In this section we show the Francy Chart Representations.

4.3.1 IsChartRep (for IsComponentObjectRep)

▷ `IsChartRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `Chart` internal representation.

4.3.2 IsChartDefaultsRep (for IsComponentObjectRep)

▷ `IsChartDefaultsRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `ChartDefaults` internal representation.

4.3.3 IsChartTypeRep (for IsComponentObjectRep)

▷ `IsChartTypeRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `ChartType` internal representation.

4.3.4 IsAxisScaleTypeRep (for IsComponentObjectRep)

▷ `IsAxisScaleTypeRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `AxisScaleType` internal representation.

4.3.5 IsAxisRep (for IsComponentObjectRep)

▷ `IsAxisRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `AxisRep` internal representation.

4.3.6 IsDatasetRep (for IsComponentObjectRep)

▷ `IsDatasetRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `DatasetRep` internal representation.

4.4 Operations

In this section we show all Francy Chart Operations.

4.4.1 Chart (for IsChartType, IsChartDefaults)

▷ `Chart(`*`IsChartType[, IsChartDefaults]`*`)` (operation)

Returns: `Chart`

Every object to draw will be a subclass of this object. This will allow all the objects to contain the same base information.

Examples:

Create a simple `Chart` of type `ChartType.BAR`:

Example
```
gap> chart:=Chart(ChartType.BAR);
gap> SetAxisXTitle(chart, "X Axis");
gap> AxisXTitle(chart);
gap> SetAxisXDomain(chart, ["domain1", "domain2", "domain3", "domain4", "domain5"]); # default []
gap> AxisXDomain(chart);
gap> SetAxisYTitle(chart, "Y Axis");
gap> AxisYTitle(chart);
gap> data1 := Dataset("data1", [100,20,30,47,90]);
gap> data2 := Dataset("data2", [51,60,72,38,97]);
gap> data3 := Dataset("data3", [50,60,70,80,90]);
gap> Add(chart, [data1, data2, data3]);
gap> Remove(chart, data1);
gap> Add(chart, data1);
gap> Remove(chart, [data2, data3]);
gap> Length(RecNames(chart!.data)) = 1;
```

Create a simple `Chart` of type `ChartType.LINE`:

Example

```
gap> chart:=Chart(ChartType.LINE);
gap> SetAxisXTitle(chart, "X Axis");
gap> SetAxisYTitle(chart, "Y Axis");
gap> data1 := Dataset("data1", [100,20,30,47,90]);
gap> Add(chart, data1);
```

Create a simple `Chart` of type `ChartType.SCATTER`:

Example

```
gap> chart:=Chart(ChartType.SCATTER);
gap> SetAxisXTitle(chart, "X Axis");
gap> SetAxisYTitle(chart, "Y Axis");
gap> data1 := Dataset("data1", [100,20,30,47,90]);
gap> Add(chart, data1);
```

4.4.2 Add (for IsChart, IsDataset)

▷ `Add(`*`IsChart[, IsDataset, List(IsDataset)]`*`)` (operation)

Returns: `Chart`

Adds a `Dataset` to a specific `Chart`.

4.4.3 Remove (for IsChart, IsDataset)

▷ `Remove(`*`IsChart[, IsDataset, List(IsDataset)]`*`)` (operation)

Returns: `Chart`

Removes a `Dataset` from a specific `Chart`.

4.4.4 Dataset (for IsString, IsList)

▷ `Dataset(`*`IsString(title), IsList(data)`*`)` (operation)

Returns: `Dataset`

Creates a dataset.

4.4.5 DefaultAxis (for IsChartType)

▷ `DefaultAxis(`*`IsChartType`*`)` (operation)

Returns: `rec`

Returns the default settings for a `ChartType`

4.4.6 XAxis (for IsAxisScaleType, IsString, IsList)

▷ `XAxis(`*`IsAxisScaleType, IsString(title), IsList(domain)`*`)` (operation)

Returns: `XAxis`

Creates a XAxis

4.4.7 YAxis (for IsAxisScaleType, IsString, IsList)

▷ `YAxis(`*`IsAxisScaleType, IsString(title), IsList(domain)`*`)` (operation)

Returns: `YAxis`

Creates a YAxis

4.5 Global

In this section we show all Global Chart Francy Records for multi purpose.

4.6 Attributes

In this section we show all Francy Attributes

4.6.1 ShowLegend (for IsChart)

▷ `ShowLegend(`*`arg`*`)` (attribute)

Returns: `IsBool` True if enabled otherwise False

`ShowLegend` is a property that enables or disables the legend in the client implementation.

4.6.2 ShowLegend (for IsChart)

▷ `ShowLegend(`*`arg1`*`)` (function)

4.6.3 SetShowLegend (for IsChart, IsBool)

▷ `SetShowLegend(`*`IsChart, IsBool`*`)` (function)

`ShowLegend` is a property that enables or disables the legend in the client implementation.

4.6.4 AxisXTitle (for IsChart)

▷ `AxisXTitle(`*`arg`*`)` (attribute)

Returns: `IsString` with the title of the object

This title is used to display the X Axis Title in the client implementation.

4.6.5 AxisXTitle (for IsChart)

▷ `AxisXTitle(`*`arg1`*`)` (function)

4.6.6 SetAxisXTitle (for IsChart, IsString)

▷ `SetAxisXTitle(`*`IsChart, IsString`*`)` (function)

This title is used to display the X Axis Title in the client implementation.

4.6.7 AxisYTitle (for IsChart)

▷ `AxisYTitle(`*`arg`*`)` (attribute)

Returns: `IsString` with the title of the object

This title is used to display the Y Axis Title in the client implementation.

4.6.8 AxisYTitle (for IsChart)

▷ `AxisYTitle(arg1)` (function)

4.6.9 SetAxisYTitle (for IsChart, IsString)

▷ `SetAxisYTitle(IsChart, IsString)` (function)

This title is used to display the Y Axis Title in the client implementation.

4.6.10 AxisXDomain (for IsChart)

▷ `AxisXDomain(arg)` (attribute)

Returns: `IsList`

This is the domain of the X Axis values in the client implementation.

4.6.11 AxisXDomain (for IsChart)

▷ `AxisXDomain(arg1)` (function)

4.6.12 SetAxisXDomain (for IsChart, IsList)

▷ `SetAxisXDomain(IsList, IsList)` (function)

This is the domain of the X Axis values in the client implementation.

Chapter 5

Francy Core

Francy is responsible for generating JSON metadata which allows external tools / libraries / frameworks to add a visual representation. This JSON representation defines the contract between this package (server) and a GUI framework (client), this enables complete SoC (Separation of Concerns). Francy can be used to provide a graphical interactive environment on existing GAP packages.

A JSON schema is present and can be used to produce clients for this package. *See schema/francy.json*

To map required / optional attributes from the schema into GAP code, the implementation follows the following criteria:

- Object creation requests mandatory attributes, i.e. required with no default value, e.g. canvas:=Canvas("Title")
- Object creation accepts an object of defaults, i.e. default values for mandatory fields but that might repeat througout the creation of multiple similar objects, e.g. defaults:=DefaultCanvas; defaults!.zoomToFit:=false; canvas:=Canvas("Title",defaults); Where DefaultCanvas contains defaults for width (800) and height (600)
- Attributes associated with the object that can be set, .i.e. optional with no defaults, e.g. canvas:=Canvas("Title"); SetTexTypesetting(canvas,true);

The API follows a common convention and adding objects to other objects is done using Add(objectHolder,object)

Although Francy has the concept of a `Graph`, it does not implement any Mathematics Graphs Theory.

Please see Francy-JS for client implementation.

5.1 Categories

In this section we show all Francy Core Categories.

5.1.1 IsFrancyObject (for IsObject)

▷ `IsFrancyObject(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies all `Objects` in Francy.

5.1.2 IsFrancyDefaultObject (for IsObject)

▷ `IsFrancyDefaultObject(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies all Default records in Francy.

5.1.3 IsFrancyTypeObject (for IsObject)

▷ `IsFrancyTypeObject(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies all Type records in Francy.

5.2 Families

In this section we show all Francy Core Families.

5.3 Global

In this section we show all Francy Core Types

5.4 Attributes

In this section we show all Francy Core Attributes

5.4.1 FrancyId (for IsFrancyObject)

▷ `FrancyId(`*arg*`)` (attribute)

Returns: `IsString` with the id of the object

All Objects created in Francy have a generated identifier.

5.4.2 FrancyId (for IsFrancyObject)

▷ `FrancyId(`*arg1*`)` (function)

Returns: `IsString` with the id of the object

Prints the object unique identifier.

5.4.3 SetFrancyId (for IsFrancyObject, IsString)

▷ `SetFrancyId(`*o, s*`)` (function)

Use with care! Changing the unique ID might be useful in certain cases, but bare in mind it might cause unexpected behavior if you're not sure about!

Chapter 6

Francy Graphs

It is possible to build `Graphs`, direct or indirect.

Please see examples section.

Please see Francy-JS for client implementation.

6.1 Categories

In this section we show all Francy Graph Categories.

6.1.1 IsFrancyGraph (for IsFrancyObject)

▷ `IsFrancyGraph(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `Graph` objects.

6.1.2 IsFrancyGraphType (for IsFrancyObject)

▷ `IsFrancyGraphType(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `GraphType` objects.

6.1.3 IsFrancyGraphDefaults (for IsFrancyDefaultObject)

▷ `IsFrancyGraphDefaults(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `GraphDefaults` objects.

6.1.4 IsShape (for IsFrancyObject)

▷ `IsShape(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `Shape` objects.

6.1.5 IsShapeType (for IsFrancyObject)

▷ `IsShapeType(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `ShapeType` objects.

6.1.6 IsShapeDefaults (for IsFrancyDefaultObject)

▷ `IsShapeDefaults(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `ShapeDefaults` objects.

6.1.7 IsLink (for IsFrancyObject)

▷ `IsLink(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `Link` objects.

6.1.8 IsLinkDefaults (for IsFrancyDefaultObject)

▷ `IsLinkDefaults(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `LinkDefaults` objects.

6.2 Families

In this section we show all Francy Graph Families.

6.3 Representations

In this section we show all Francy Graph Representations.

6.3.1 IsFrancyGraphRep (for IsComponentObjectRep)

▷ `IsFrancyGraphRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `Graph` internal representation.

6.3.2 IsFrancyGraphDefaultsRep (for IsComponentObjectRep)

▷ `IsFrancyGraphDefaultsRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `GraphDefaults` internal representation.

6.3.3 IsFrancyGraphTypeRep (for IsComponentObjectRep)

▷ `IsFrancyGraphTypeRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `GraphType` internal representation.

6.3.4 IsShapeRep (for IsComponentObjectRep)

▷ `IsShapeRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `Shape` internal representation.

6.3.5 IsShapeDefaultsRep (for IsComponentObjectRep)

▷ `IsShapeDefaultsRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `ShapeDeafults` internal representation.

6.3.6 IsShapeTypeRep (for IsComponentObjectRep)

▷ `IsShapeTypeRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `ShapeType` internal representation.

6.3.7 IsLinkRep (for IsComponentObjectRep)

▷ `IsLinkRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `Link` internal representation.

6.3.8 IsLinkDefaultsRep (for IsComponentObjectRep)

▷ `IsLinkDefaultsRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `LinkDeafults` internal representation.

6.4 Operations

In this section we show all Francy Graph Operations.

6.4.1 Graph (for IsFrancyGraphType, IsFrancyGraphDefaults)

▷ `Graph(`*IsFrancyGraphType[, IsFrancyGraphDefaults]*`)` (operation)

Returns: `Graph`

Every object to draw will be a subclass of this object. This will allow all the objects to contain the same base information.

Examples:

Create a simple `Graph` of type `GraphType.DIRECTED` and simple `Shapes` connected with `Links`:

```
Example
gap> graph := Graph(GraphType.DIRECTED);
gap> SetSimulation(graph, false);
gap> shape1 := Shape(ShapeType.SQUARE);
gap> shape1!.layer := 1;
gap> shape2 := Shape(ShapeType.TRIANGLE);
gap> shape2!.layer := 3;
gap> link := Link(shape1, shape2);
gap> Add(graph, link);
gap> Add(graph, [shape1, shape2]);
```

Create a simple Graph of type GraphType.UNDIRECTED and a simple Shape with a TriggerEvent.RIGHT_CLICK Callback:

```
Example
gap> graph := Graph(GraphType.UNDIRECTED);
gap> shape := Shape(ShapeType.SQUARE);
gap> MyFunction := function() Add(graph, Shape(ShapeType.Circle)); return graph; end;
gap> callback := Callback(TriggerType.RIGHT_CLICK, MyFunction);
gap> Add(shape, callback);
gap> Add(graph, shape);
```

6.4.2 UnsetNodes (for IsFrancyGraph)

▷ UnsetNodes(*arg*) (operation)

Removes all nodes from gaph

6.4.3 UnsetLinks (for IsFrancyGraph)

▷ UnsetLinks(*arg*) (operation)

Removes all nodes from gaph

6.4.4 Add (for IsFrancyGraph, IsLink)

▷ Add(*IsFrancyGraph[, IsLink, List(IsLink)]*) (operation)

Returns: Graph

Add IsLink to a specific Graph.

6.4.5 Remove (for IsFrancyGraph, IsLink)

▷ Remove(*IsFrancyGraph[, IsLink, List(IsLink)]*) (operation)

Returns: Graph

Remove IsLink from a specific Graph.

6.4.6 Add (for IsFrancyGraph, IsShape)

▷ Add(*IsFrancyGraph[, IsShape, List(IsShape)]*) (operation)

Returns: Graph

Add IsShape to a specific Graph.

6.4.7 Remove (for IsFrancyGraph, IsShape)

▷ `Remove(`*`IsFrancyGraph[, IsShape, List(IsShape)]`*`)` (operation)

Returns: `Graph`

Remove `IsShape` from a specific `Graph`.

6.4.8 Shape (for IsShapeType, IsString, IsShapeDefaults)

▷ `Shape(`*`IsShapeType[, IsString(title), IsShapeDefaults]`*`)` (operation)

Returns: `Shape`

Every object to draw will be a subclass of this object. This will allow all the objects to contain the same base information.

6.4.9 GetShape (for IsFrancyGraph, IsString)

▷ `GetShape(`*`IsFrancyGraph, IsString`*`)` (operation)

Returns: `Shape`

Gets a `Shape` node from a graph by ID.

6.4.10 GetShapes (for IsFrancyGraph)

▷ `GetShapes(`*`IsFrancyGraph, IsString`*`)` (operation)

Returns: `List(Shape)`

Gets a `Shape` node from a graph by ID.

6.4.11 Add (for IsShape, IsMenu)

▷ `Add(`*`IsShape[, IsMenu, List(IsMenu)]`*`)` (operation)

Returns: `Shape`

Add `Menu` to a specific `Shape`.

6.4.12 Remove (for IsShape, IsMenu)

▷ `Remove(`*`IsShape[, IsMenu, List(IsMenu)]`*`)` (operation)

Returns: `Shape`

Remove `Menu` from a specific `Shape`.

6.4.13 Add (for IsShape, IsCallback)

▷ `Add(`*`IsShape[, IsCallback, List(IsCallback)]`*`)` (operation)

Returns: `Shape`

Add `Callback` to a specific `Shape`.

6.4.14 Remove (for IsShape, IsCallback)

▷ `Remove(`*`IsShape[, IsCallback, List(IsCallback)]`*`)` (operation)

Returns: `Shape`

Remove `Callback` from a specific `Shape`.

6.4.15 Add (for IsShape, IsFrancyMessage)

▷ `Add(`*`IsShape[, IsFrancyMessage, List(IsFrancyMessage)]`*`)` (operation)

Returns: `Shape`

Add `Callback` to a specific `Shape`.

6.4.16 Remove (for IsShape, IsFrancyMessage)

▷ `Remove(`*`IsShape[, IsFrancyMessage, List(IsFrancyMessage)]`*`)` (operation)

Returns: `Shape`

Remove `Callback` from a specific `Shape`.

6.4.17 Link (for IsShape, IsShape, IsLinkDefaults)

▷ `Link(`*`IsShape, IsShape`*`)` (operation)

Returns: `Link`

Creates a `Link` between the two `Shape`.

6.4.18 Links (for IsList, IsList, IsLinkDefaults)

▷ `Links(`*`List(IsShape), List(IsShape)`*`)` (operation)

Returns: `List(Link)`

Creates a `Link` between the `Shape` of the first list and the second list.

6.4.19 GetLink (for IsFrancyGraph, IsString)

▷ `GetLink(`*`IsFrancyGraph, IsString`*`)` (operation)

Returns: `Link`

Gets a `Link` from a graph by ID.

6.4.20 GetLinks (for IsFrancyGraph)

▷ `GetLinks(`*`IsFrancyGraph, IsString`*`)` (operation)

Returns: `List(Link)`

Gets a `Link` from a graph.

6.5 Global

In this section we show all Global Callback Francy Records for multi purpose.

6.6 Attributes

In this section we show all Francy Core Attributes

6.6.1 Title (for IsShape)

▷ `Title(`*arg*`)` (attribute)

Returns: `IsString` with the title of the object

Sets the title on the Shape. Supports LaTex syntax that gets translated, if enabled on the client.

6.6.2 Title (for IsShape)

▷ `Title(`*arg1*`)` (function)

6.6.3 SetTitle (for IsShape, IsString)

▷ `SetTitle(`*IsRequiredArg, IsString*`)` (function)

Sets the title of the Shape.

6.6.4 Color (for IsShape)

▷ `Color(`*arg*`)` (attribute)

Returns: `IsInt`

The `Color` of the current shape.

6.6.5 Color (for IsShape)

▷ `Color(`*arg1*`)` (function)

6.6.6 SetColor (for IsShape, IsString)

▷ `SetColor(`*IsShape, IsString*`)` (function)

Sets the `Color` value.

6.6.7 PosX (for IsShape)

▷ `PosX(`*arg*`)` (attribute)

Returns: `IsInt`

The Position in the X Axis of the Shape in the Canvas in pixels

6.6.8 PosX (for IsShape)

▷ `PosX(`*arg1*`)` (function)

6.6.9 SetPosX (for IsShape, IsInt)

▷ `SetPosX(`*`IsShape, IsInt`*`)` (function)

Sets the Position in the X Axis of the Shape in the Canvas in pixels

6.6.10 PosY (for IsShape)

▷ `PosY(`*`arg`*`)` (attribute)

Returns: `IsInt`

The Position in the Y Axis of the Shape in the Canvas in pixels

6.6.11 PosY (for IsShape)

▷ `PosY(`*`arg1`*`)` (function)

6.6.12 SetPosY (for IsShape, IsInt)

▷ `SetPosY(`*`IsShape, IsInt`*`)` (function)

Sets the Position in the Y Axis of the Shape in the Canvas in pixels

6.6.13 Size (for IsShape)

▷ `Size(`*`arg`*`)` (attribute)

Returns: `IsPosInt`

The `Size` of the Shape

6.6.14 Size (for IsShape)

▷ `Size(`*`arg1`*`)` (function)

6.6.15 SetSize (for IsShape, IsPosInt)

▷ `SetSize(`*`IsShape, IsPosInt`*`)` (function)

Sets the Size of the Shape

6.6.16 Layer (for IsShape)

▷ `Layer(`*`arg`*`)` (attribute)

Returns: `IsInt`

The `Layer` in which the node will be placed. This property is also used to apply a color based on a scale

6.6.17 Layer (for IsShape)

▷ `Layer(`*`arg1`*`)` (function)

6.6.18 SetLayer (for IsShape, IsInt)

▷ `SetLayer(`*`IsShape, IsInt`*`)` (function)

Sets the `Layer` number.

6.6.19 ParentShape (for IsShape)

▷ `ParentShape(`*`arg`*`)` (attribute)

Returns: `IsShape`

The `ParentShape` in which the node will be placed. This property is also used to apply a color based on a scale

6.6.20 ParentShape (for IsShape)

▷ `ParentShape(`*`arg1`*`)` (function)

6.6.21 SetParentShape (for IsShape, IsShape)

▷ `SetParentShape(`*`IsShape, IsShape`*`)` (function)

Sets the `ParentShape`.

6.6.22 Simulation (for IsFrancyGraph)

▷ `Simulation(`*`arg`*`)` (attribute)

Returns: `IsBool` True if enabled otherwise False

`Simulation` is a property that sets the simulation behavior by applying forces to organize the graphics, without the need to provide custom positions, in the client implementation.

6.6.23 Simulation (for IsFrancyGraph)

▷ `Simulation(`*`arg1`*`)` (function)

6.6.24 SetSimulation (for IsFrancyGraph, IsBool)

▷ `SetSimulation(`*`IsCanvas, IsBool`*`)` (function)

Sets the `Simulation` behavior.

6.6.25 Collapsed (for IsFrancyGraph)

▷ `Collapsed(`*arg*`)` (attribute)

Returns: `IsBool` True if enabled otherwise False

`Collapsed` is a property that sets to collapsed the graphic structure by default

6.6.26 Collapsed (for IsFrancyGraph)

▷ `Collapsed(`*arg1*`)` (function)

6.6.27 SetCollapsed (for IsFrancyGraph, IsBool)

▷ `SetCollapsed(`*IsCanvas, IsBool*`)` (function)

Sets the `Collapsed` behavior.

6.6.28 Selected (for IsShape)

▷ `Selected(`*arg*`)` (attribute)

Returns: `IsBool` True if enabled otherwise False

`Collapsed` is a property that sets to collapsed the graphic structure by default

6.6.29 Selected (for IsShape)

▷ `Selected(`*arg1*`)` (function)

6.6.30 SetSelected (for IsShape, IsBool)

▷ `SetSelected(`*IsCanvas, IsBool*`)` (function)

Sets the `Collapsed` behavior.

6.6.31 ConjugateId (for IsShape)

▷ `ConjugateId(`*arg*`)` (attribute)

Returns: `IsBool` True if enabled otherwise False

`Collapsed` is a property that sets to collapsed the graphic structure by default

6.6.32 ConjugateId (for IsShape)

▷ `ConjugateId(`*arg1*`)` (function)

6.6.33 SetConjugateId (for IsShape, IsInt)

▷ `SetConjugateId(`*`IsCanvas, IsBool`*`)` (function)

Sets the `Collapsed` behavior.

6.6.34 Weight (for IsLink)

▷ `Weight(`*`arg`*`)` (attribute)

Returns: `IsInt`

The `Weight` of the current link.

6.6.35 Weight (for IsLink)

▷ `Weight(`*`arg1`*`)` (function)

6.6.36 SetWeight (for IsLink, IsInt)

▷ `SetWeight(`*`IsLink, IsInt`*`)` (function)

Sets the `Weight` value.

6.6.37 Length (for IsLink)

▷ `Length(`*`arg`*`)` (attribute)

Returns: `IsInt`

The `Length` of the current link.

6.6.38 Length (for IsLink)

▷ `Length(`*`arg1`*`)` (function)

6.6.39 SetLength (for IsLink, IsInt)

▷ `SetLength(`*`IsLink, IsInt`*`)` (function)

Sets the `Length` value.

6.6.40 Invisible (for IsLink)

▷ `Invisible(`*`arg`*`)` (attribute)

Returns: `IsBoolean`

The `Invisible` of the current link.

6.6.41 Invisible (for IsLink)

▷ `Invisible(`*`arg1`*`)` (function)

6.6.42 SetInvisible (for IsLink, IsBool)

▷ `SetInvisible(`*`IsLink, IsBool`*`)` (function)

Sets the `Invisible` value.

6.6.43 Color (for IsLink)

▷ `Color(`*`arg`*`)` (attribute)

Returns: `IsInt`

The `Color` of the current link.

6.6.44 Color (for IsLink)

▷ `Color(`*`arg1`*`)` (function)

6.6.45 SetColor (for IsLink, IsString)

▷ `SetColor(`*`IsShape, IsString`*`)` (function)

Sets the `Color` value.

6.6.46 Title (for IsLink)

▷ `Title(`*`arg`*`)` (attribute)

Returns: `IsInt`

The `Title` of the current link.

6.6.47 Title (for IsLink)

▷ `Title(`*`arg1`*`)` (function)

6.6.48 SetTitle (for IsLink, IsString)

▷ `SetTitle(`*`IsShape, IsString`*`)` (function)

Sets the `Title` value.

Chapter 7

Francy Menus

Menus are agregators of actions that are represented here by `Callbacks`. Menus can have SubMenus, and are constituted by a Title and a Callback.

Please see Francy-JS for client implementation.

7.1 Categories

In this section we show all Francy Menu Categories.

7.1.1 IsMenu (for IsFrancyObject)

▷ `IsMenu(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `Menu` objects.

7.2 Families

In this section we show all Francy Menu Families.

7.3 Representations

In this section we show all Francy Menu Representations.

7.3.1 IsMenuRep (for IsComponentObjectRep)

▷ `IsMenuRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `Menu` internal representation.

7.4 Operations

In this section we show all Francy Menu Operations.

7.4.1 Menu (for IsString, IsCallback)

▷ `Menu(`*`IsString(title)[, IsCallback]`*`)` (operation)

Returns: `Menu`

Creates a Menu for a `Callback` Is up to the client implementation to sort out the Menu and invoke the `Callback`

7.4.2 Add (for IsMenu, IsMenu)

▷ `Add(`*`IsMenu[, IsMenu, List(IsMenu)]`*`)` (operation)

Returns: `Menu`

Add `Menu` to a specific `Menu` creating a Submenu. Is up to the client implementation to handle this.

7.4.3 Remove (for IsMenu, IsMenu)

▷ `Remove(`*`IsMenu[, IsMenu, List(IsMenu)]`*`)` (operation)

Returns: `Menu`

Remove `Menu` from a specific `Menu`. The client should be able to handle this.

7.5 Attributes

In this section we show all Francy Core Attributes

7.5.1 Title (for IsMenu)

▷ `Title(`*`arg`*`)` (attribute)

Returns: `IsString` with the title of the object

A title on a `Menu` is used to identify the menu entry.

7.5.2 Title (for IsMenu)

▷ `Title(`*`arg1`*`)` (function)

7.5.3 SetTitle (for IsMenu, IsString)

▷ `SetTitle(`*`IsMenu, IsString`*`)` (function)

Sets the title of the `Menu`.

Chapter 8

Francy Messages

`FrancyMessage` is an object that holds a message.

These messages can be used to provide information to users in the form of `SUCCESS`, `INFO`, `WARNING`, `ERROR`. Please see Francy-JS for client implementation.

8.1 Categories

In this section we show all Francy FrancyMessage Categories.

8.1.1 IsFrancyMessage (for IsFrancyObject)

▷ `IsFrancyMessage(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `FrancyMessage` objects.

8.1.2 IsFrancyMessageType (for IsFrancyObject)

▷ `IsFrancyMessageType(`*arg*`)` (filter)

Returns: `true` or `false`

Identifies `MessageType` objects.

8.2 Families

In this section we show all Francy FrancyMessage Families.

8.3 Representations

In this section we show all Francy FrancyMessage Representations.

8.3.1 IsFrancyMessageRep (for IsComponentObjectRep)

▷ `IsFrancyMessageRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `FrancyMessage` internal representation.

8.3.2 IsFrancyMessageTypeRep (for IsComponentObjectRep)

▷ `IsFrancyMessageTypeRep(`*arg*`)` (filter)

Returns: `true` or `false`

Checks whether an `Object` has a `FrancyMessage` internal representation.

8.4 Operations

In this section we show all Francy FrancyMessage Operations.

8.4.1 FrancyMessage (for IsFrancyMessageType, IsString, IsString)

▷ `FrancyMessage(`*IsString, IsString*`)` (operation)

Returns: `FrancyMessage`

Adds an info label with the format label: value

8.5 Global

In this section we show all Global Callback Francy Records for multi purpose.

8.6 Attributes

In this section we show all Francy Core Attributes

8.6.1 Title (for IsFrancyMessage)

▷ `Title(`*arg*`)` (attribute)

Returns: `IsString` with the title of the object

A title on a `FrancyMessage` is used to display the title information to the user.

8.6.2 Title (for IsFrancyMessage)

▷ `Title(`*arg1*`)` (function)

8.6.3 SetTitle (for IsFrancyMessage, IsString)

▷ `SetTitle(`*IsFrancyMessage, IsString*`)` (function)

Sets the title of the `FrancyMessage`.

8.6.4 Value (for IsFrancyMessage)

▷ `Value(`*arg*`)` (attribute)

Returns: `IsString` with the title of the object

A value on a `FrancyMessage` is used to display the information to the user.

8.6.5 Value (for IsFrancyMessage)

▷ `Value(arg1)` (function)

8.6.6 SetValue (for IsFrancyMessage, IsString)

▷ `SetValue(IsFrancyMessage, IsString)` (function)

Sets the actual message of the `FrancyMessage`.

Chapter 9

Francy Util

9.1 Operations

In this section we show all Francy Util Operations. Contains utility methods to handle Object printing/viewing, Sanitizing, etc.

9.1.1 JUPYTER_ViewString (for IsObject)

▷ `JUPYTER_ViewString(`*`arg`*`)` (operation)

Returns: `String`

This method will pretty print in jupyter environment.

9.1.2 Sanitize (for IsObject)

▷ `Sanitize(`*`IsObject`*`)` (operation)

Returns: `rec`

This method will clone a `Object` and return a sanitized record, traversing all the components and sanitizing when appropriate. Sanitizing in this context means, replace everything with it's string representation that can't be converted into JSON!

9.1.3 MergeObjects (for IsFrancyObject, IsFrancyObject)

▷ `MergeObjects(`*`IsFrancyObject, IsFrancyObject`*`)` (operation)

Returns: `rec`

This method will merge the properties of 2 `IsFrancyObjects` into one `rec`.

9.1.4 GenerateID

▷ `GenerateID()` (operation)

Returns: `IsString`

This method will generate a sequential ID for use as object identifier.

Index