



USER MANUAL

© 2024 GDK Software

www.codolex.com

1. Introduction	7
1.1 The idea behind codolex	8
2. Getting started with Codolex	9
2.1 Installation	10
2.2 IDE integration	10
2.3 Your first Codolex Application	11
2.3.1 Create a new project	12
2.3.2 Add a module	13
2.3.3 Editing flows	15
2.3.4 Using flows in code	16
2.3.5 Adding a datasource	17
2.3.6 Using entities	18
2.4 Codolex Concepts	20
2.4.1 Project explorer	20
2.4.2 Modules	20
2.4.3 Flow classes	21
2.4.4 Flows	22
2.4.5 Data sources	22
2.4.6 Local data sources	22
2.4.7 Entities	22
2.4.8 Search	23
3. Code generation	25
3.1 Automatic code generation	26
3.2 Manual code generation	26
3.3 File structure	27
4. Flows	28
4.1 Flow handling	29
4.2 Start and End	31
4.3 Flow results	31
4.4 Parameters	32
4.5 Visibility	32
4.6 Code preview	33
4.7 Validation	33
5. Codolex API	35
5.1 API Setup	36
5.2 Exposing Data	38
5.3 Adding endpoints with flows	39

5.4	Configure security	43
6.	Data sources	46
6.1	Local data sources	47
6.2	Connecting to databases	48
6.3	Custom database connections	49
6.3.1	Nexus DB	49
6.4	Entities	49
6.5	Nullable fields	50
6.6	Importing data structures	51
6.7	IniFile Datasource	51
6.8	JSON datasource	52
6.9	Plugin datasources	55
7.	Activities	56
7.1	Structural	57
7.1.1	Parameters	57
7.1.2	End	59
7.1.3	Start	60
7.1.4	Sequences	61
7.1.5	Decision	61
7.1.6	Merge	62
7.1.7	Loop	63
7.1.8	Exceptions	65
7.2	Core	66
7.2.1	Flow call	67
7.2.2	Code snippet	68
7.2.3	Use unit	68
7.3	Clipboard	69
7.3.1	Write to clipboard	69
7.3.2	Read from clipboard	70
7.3.3	Clear clipboard	71
7.4	Database	71
7.4.1	Get from DB	72
7.4.2	Save object	74
7.4.3	Delete object	74
7.4.4	Transactions	75
7.4.4.1	Start transaction	76
7.4.4.2	Commit transaction	76
7.4.4.3	Rollback transaction	77
7.4.5	Execute command	78
7.5	Date/Time	79

7.5.1	Calculations	80
7.5.2	Check	82
7.5.3	Conversion	83
7.5.4	Decode	85
7.5.5	Encode	86
7.5.6	Format	87
7.5.7	From	88
7.5.8	Operation	89
7.5.9	Utils	91
7.5.10	Validation	92
7.6	Dialogs	95
7.6.1	OpenDialog	95
7.6.2	SaveDialog	97
7.6.3	ShowDialog	98
7.6.4	Show modal form	99
7.7	Encoding	100
7.7.1	Decode	100
7.7.2	Encode	101
7.8	Entity conversion	102
7.8.1	Entity to JSON	103
7.8.2	JSON to entity	104
7.8.3	Entity to key/value	106
7.8.4	Key/value to entity	106
7.9	File system	107
7.9.1	Copy file/folder	108
7.9.2	Create file/folder	109
7.9.3	Delete file/folder	111
7.9.4	Exists file/folder	111
7.9.5	Get path part	112
7.9.6	Get system path	115
7.9.7	Listing file/folder	116
7.9.8	Move file/folder	116
7.9.9	Path validations	118
7.9.10	Read file	123
7.9.11	Write file	123
7.10	Hashing	124
7.10.1	Hash file	125
7.10.2	Hash string/bytes	126
7.11	Import/Export	127
7.11.1	CSV export	127
7.11.2	CSV import	128
7.12	IniFile	130
7.12.1	IniFile Read	130

7.12.2	IniFile Write	131
7.13	JSON	132
7.13.1	Text to JSON	133
7.13.2	JSON to text	134
7.13.3	Get JSON value	134
7.14	Math	136
7.14.1	Calculation	136
7.14.2	Checks	140
7.14.3	Finance	143
7.14.4	Rounding	148
7.15	Rest operation	150
7.16	Regular expressions	155
7.16.1	Options	156
7.16.2	Escape chars	156
7.16.3	IsMatch	157
7.16.4	Split string	157
7.16.5	Search match	158
7.16.6	Replace match	159
7.17	String utils	160
7.17.1	String change	161
7.17.2	String check	164
7.17.3	String conversion	166
7.17.4	String find	169
7.17.5	String parts	170
7.17.6	String split	173
7.18	Variables	174
7.18.1	Create variable	174
7.18.2	Clone entity	177
7.18.3	Change variable	178
7.18.4	List Operations	178
7.18.5	Copy entity data	180
7.18.6	Free Object	181
7.18.7	Get by association	182
7.19	Zipping	184
7.19.1	Create/open zip file	184
7.19.2	Extract zipfile	185
7.19.3	Add to zipfile	186
7.19.4	Close zipfile	188
7.19.5	Listing zip	188
7.19.6	Read zip	189
7.20	Advanced	190
7.20.1	OAuth2	190
7.21	OpenAI	193

7.21.1	AI Chat	193
8.	Creating your own activity	196
8.1	Getting started	197
8.2	Implementation	199
8.3	Tags	207
8.4	Defined entity	208
8.5	CodeGen	211
8.6	Validation	213
8.7	Testing	215
8.8	SynEdit Downloads	218
9.	Command line interface	219
10.	Best practices	221
10.1	Source control	222
10.2	Useful tips and tricks	222
10.3	Use multiple flows	222
11.	FAQ	223
12.	Release notes	225
12.1	Version 2.8.0	226
12.2	Version 2.7.0	230
12.2.1	Version 2.7.1	231
12.3	Version 2.6.0	232
12.4	Version 2.5.0	235
12.5	Version 2.4.0	241
12.5.1	Version 2.4.1	244
12.6	Version 2.3.0	244
12.7	Version 2.2.1	248
12.8	Version 2.1.0	252
Index		0

Introduction

1 Introduction



Welcome to the official guide for Codolex. Here, you will find all the information you need to get started with Codolex. [Click here](#) to open the latest version of the documentation online.

In the case that you cannot find what you are looking for, please contact support via support@codolex.com.

1.1 The idea behind codolex

Codolex is a Low Code tool designed for Delphi developers. Codolex enables you to develop business logic in a visual way. The primary objective of Codolex is to simplify the development process for every Delphi developer. Using Codolex, you can create applications by using flows and building blocks or activities. These activities can be low-level, such as reading a file, or high-level, such as linking directly to a REST Service. Codolex also simplifies working with databases and data while keeping the business logic insightful. Finally, Codolex generates code based on flows, which eliminates vendor lock-in and provides access to the source code.

Codolex turns your visual flows into platform independent code. This means that the code generated by Codolex can work on any platform where Delphi is supported.

Getting started with Codolex

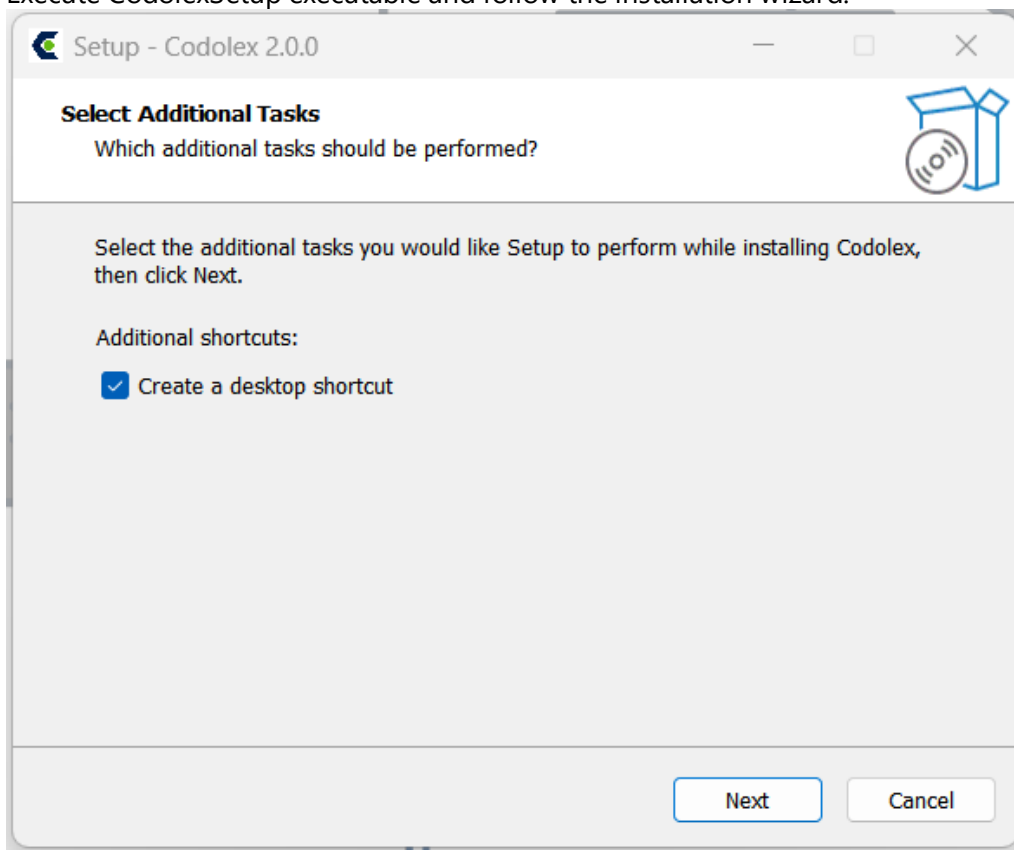
2 Getting started with Codolex

To run Codolex you need to have a recent version of Delphi installed on your environment (Delphi 10.3, 10.4, 11.x or later). Codolex supports Windows 10 or later.

Download the corresponding version of Codolex for your Delphi installation here: <https://codolex.com/download>

2.1 Installation

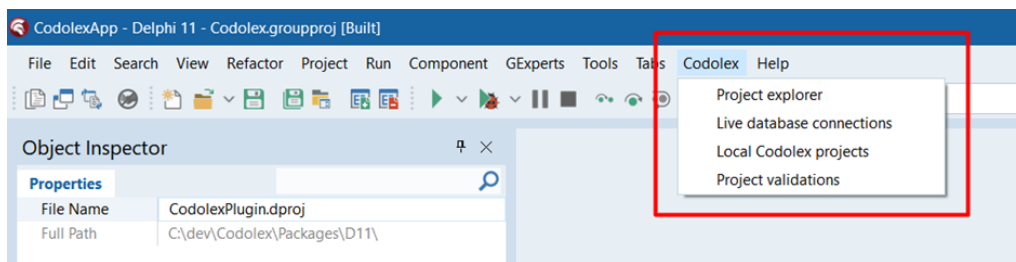
The Codolex software installation is easy and straightforward. Execute CodolexSetup executable and follow the installation wizard.



There are three ways to use Codolex, using the Delphi IDE plugin, using the standalone version, or using the command line compiler. We'll cover the IDE integration first.

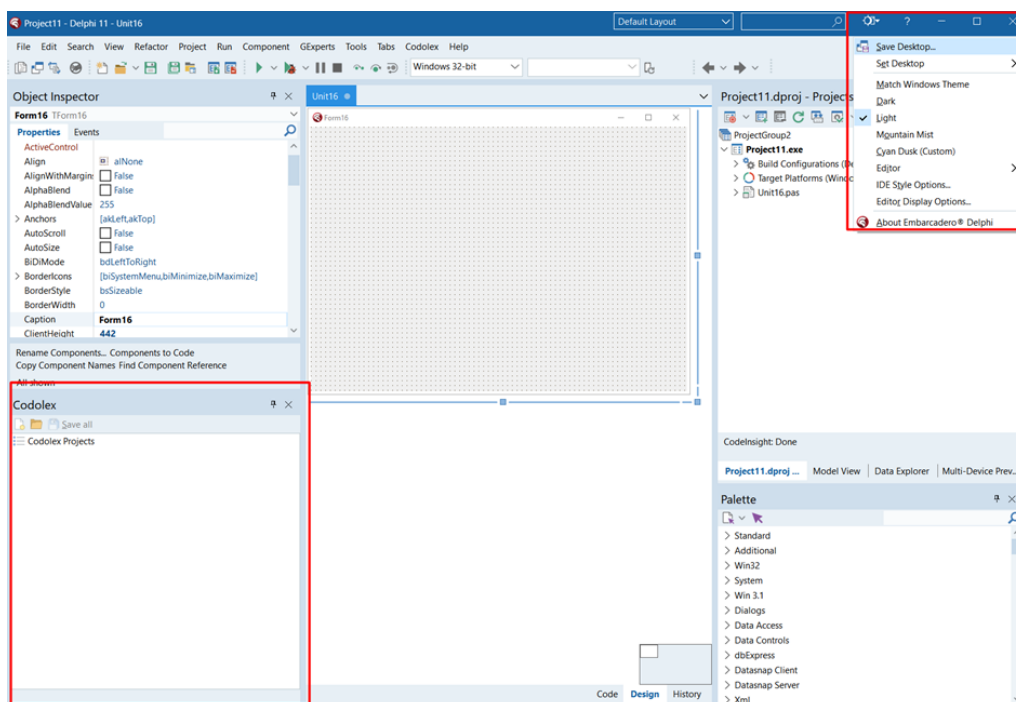
2.2 IDE integration

The Codolex installation will add a menu to the Delphi IDE.



The project explorer is the main window where you can see Codorex projects and the Codorex structure. You can create a new Codorex project or open an existing one from the project explorer, which is linked to your active Delphi project. When you compile your Delphi project, Codorex first generates the source code automatically, and adds the required paths to the search path of your Delphi project.

To make the integration work as smoothly as possible, it is best to 'dock' the project explorer within the Delphi IDE. When you save the layout, the project explorer remains in the same place, even if you restart Delphi.



2.3 Your first Codorex Application

In your first Codorex application, you will build a simple Delphi application to consume REST services and develop business logic to work with the results of the REST calls. We will create an example application to query a time API to get the current time of a specified time zone.

this guide follows the next 6 steps:

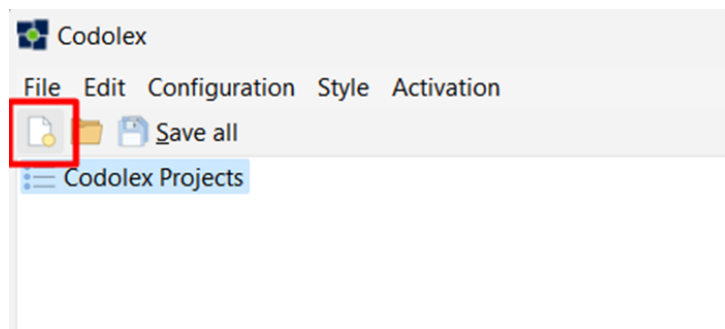
- [Create a new project](#)¹²
- [Add a module](#)¹³
- [Editing flows](#)¹⁵
- [Using flows in code](#)¹⁶
- [Adding a datasource](#)¹⁷
- [Using entities](#)¹⁸

Follow these steps if you want to learn more about the delphi low-code tool and create your first application.

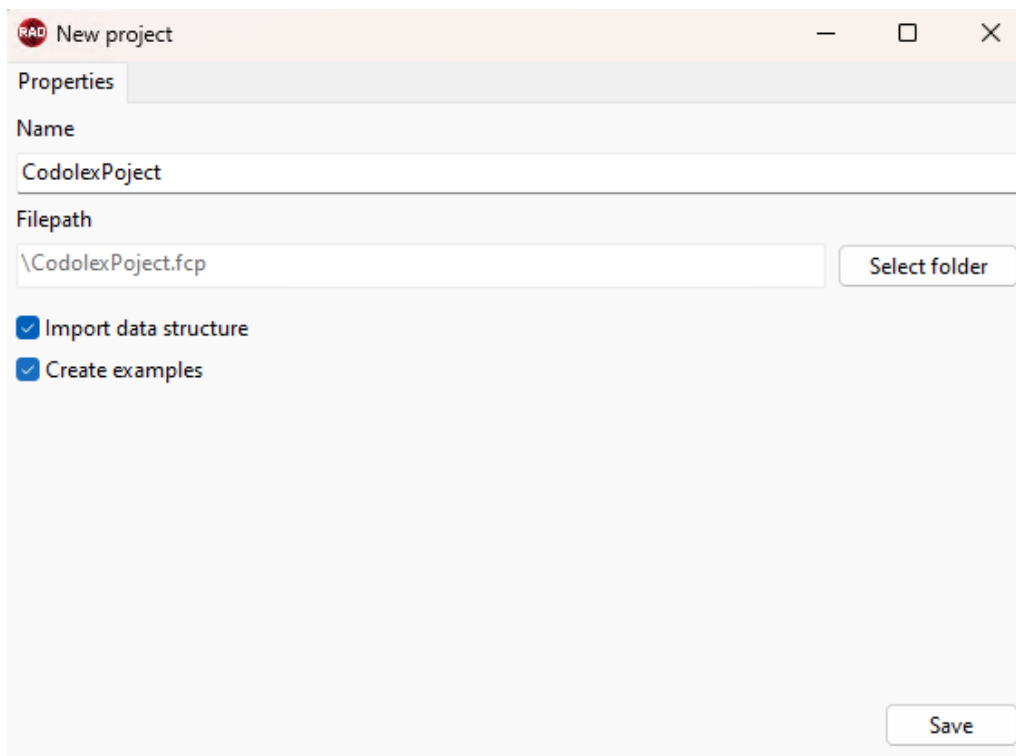
If you are already comfortable with the basic, check the [Activities](#)⁵⁷ or other topic to learn about all the possibilities of Codolex.

2.3.1 Create a new project

To start a new Codolex project, first create an empty Delphi application (VCL or FireMonkey), save your empty Delphi project, and click the "new project" button in the Codolex project explorer form:



This will open up the project wizard. The wizard helps in setting up your first project.



The first step is to choose a name for the project and the second step is to choose a location for the project.

A recommended location is in a subfolder next to the project file with the name 'Codolex'.

The third step is to import a data structure.

Data structures are the key to making use of codolex. It maps your database or JSON structure into entities that codolex can use in activities.

You can read more on how to import a data structure in the topic [Adding a datasource](#)^[17], or continue the guide for your first application by deselection the option to import

At last you have the option to let codolex create some examples for you. This will create a basic structure of flows to use and change. For this guide it's not needed, but you can always use the examples and explore Codolex your way.

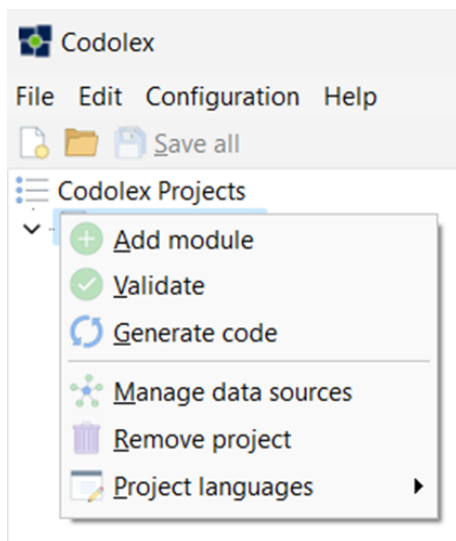
If you want to follow the guide go to the next topic: [Add a module](#)^[13]

2.3.2 Add a module

previous topic: [Create a new project](#)^[12]

Next, we are going to create a module. Think of a module as a way to group business logic on a high level, similar to a folder in the Delphi project structure.

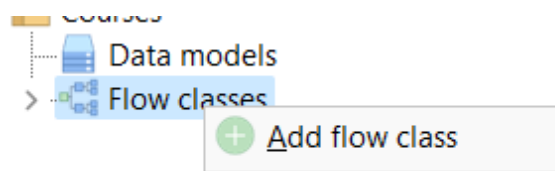
Right-click on the project and choose "Add module".



Enter a name for the module, for example TimeAPI, and click "Save". A module with this name will now be visible in the project explorer.

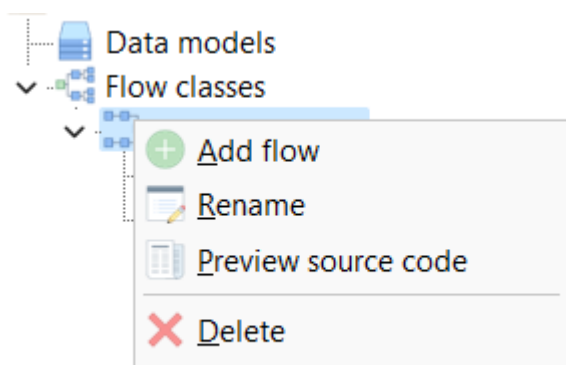
Now create a flow class. Codolux flow classes have the same purpose and meaning as classes in code. Within a class you can define private, protected and public methods. These are known as "flows".

Create a flow class by right clicking on the "Flow classes" item below your opened module.

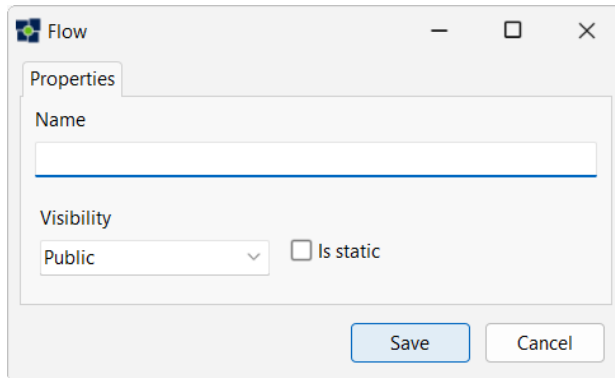


Give a name for the flow class, for example TimeZone, and click "Save". The flow class will appear in the project explorer. It is not necessary to use the prefix "T" for a flowclass. Codolux will add the "T" when the source code is generated.

Now create a flow. The flow is the equivalent of a procedure or function in a class. Add a flow by right clicking on your flow class in the project explorer. Choose "Add flow".



In the next form, choose the name, for example GetTimeOfTimeZone. Keep the visibility of your flow public, and click "Save". We will cover the options shown here later.



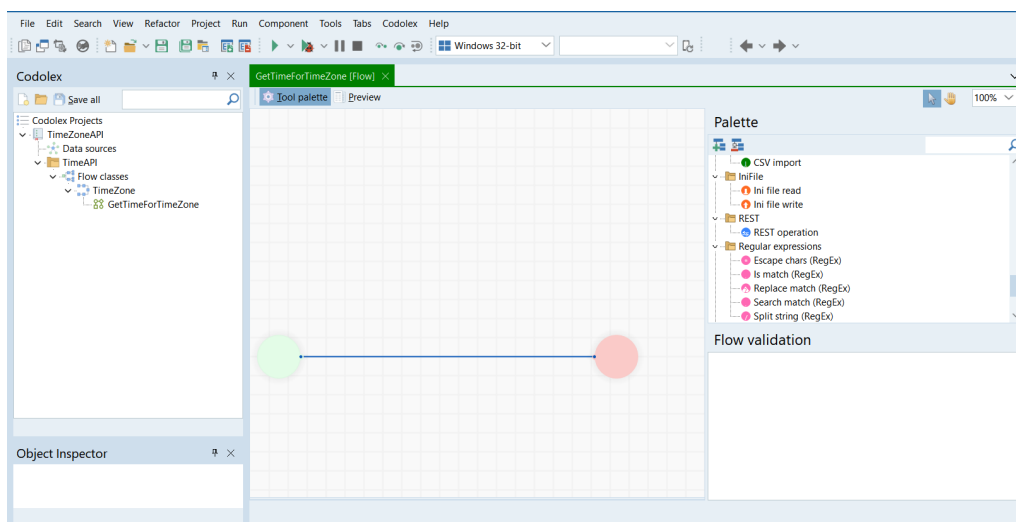
The Codolex flow editor will open.

Go to the next topic for editing flows. Where we will build the business logic to connect to the timezone api.

[Editing flows](#) ¹⁵

2.3.3 Editing flows

previous topic: [Add a module](#) ¹³



To consume a REST services, drop a REST Operation activity on the flow. You can do this by selecting the REST Operation activity in the Codolex palette, and drag this on the line between the start of the flow (the green dot) and the end of the flow (the red dot).

Next, double click on the REST Operation activity in the flow.

For this first example, we just use the following URL to retrieve the JSON from the Rest API:

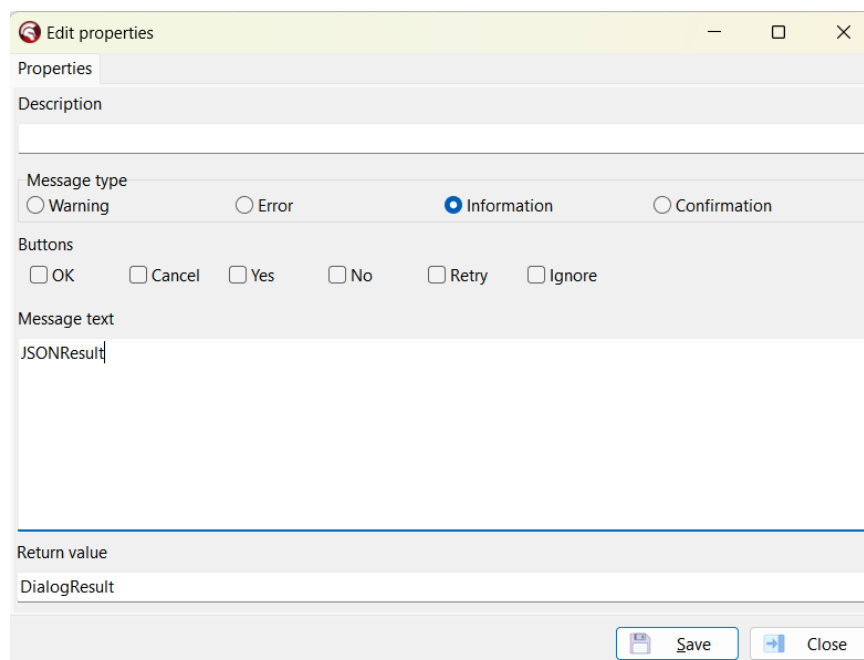
<https://timeapi.io/api/Time/current/zone?timeZone=Europe/Amsterdam>

Copy and paste this URL in the URL edit box. Next, click the three dots next to the edit box, and click stringify (alternatively, use CTRL+D). In this example, we need to make sure the URL's content field content is a string so we need to add quotation marks around it to specify this as actual text. It is also possible to use a variable in the URL field and in that case, no quotation marks would be needed.

Change the name of the result text to JsonResult, and click Save.

If you want, you can now check the code by clicking the Preview button at the top of the flow screen to see the code that Codolux will generate.

To show the result of this call, let's drag and drop a ShowDialog activity after the REST Operation activity, open it, put JsonResult as the text and set the message type to Information.



The next topic will show how to use the flow in our delphi project

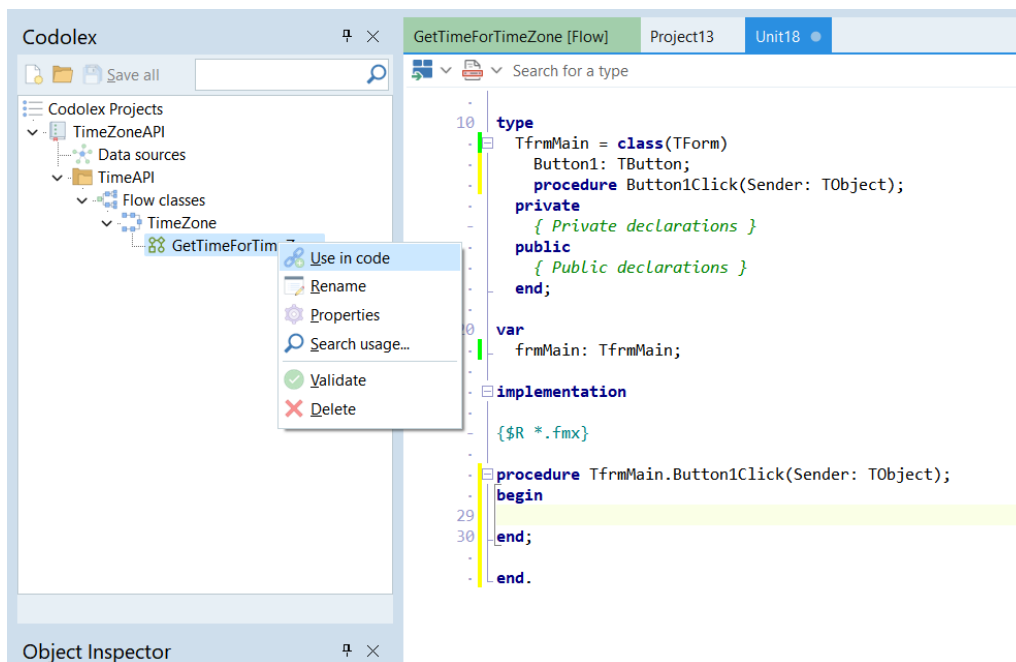
[Using flows in code](#)¹⁶

2.3.4 Using flows in code

Previous topic: [Editing flows](#)¹⁵

The last thing we have to do is to call this flow from your Delphi application. To do this, go to your Delphi main form and drop a TButton on the form. Double click on the button to create the OnClick event.

To execute the flow, right click on the GetTimeFromTimeZone flow, and click "Use in code".



You will notice that Codorex created the Flow class, added the call to the flow, and added the required uses to the unit.

```
uses
    TimeZoneAPI.TimeAPI.TimeZone, TimeZoneAPI.TimeAPI.TimeZone.Interfaces;

{$R *.fmx}

procedure TfrmMain.Button1Click(Sender: TObject);
begin
    var TimeZone: ITimeZone := TTimeZone.Create;
    TimeZone.GetTimeForTimeZone();
end;
```

Run the program and click on the button to see the result of the REST Operation.

Next stop:

[Adding a datasource](#) ¹⁷

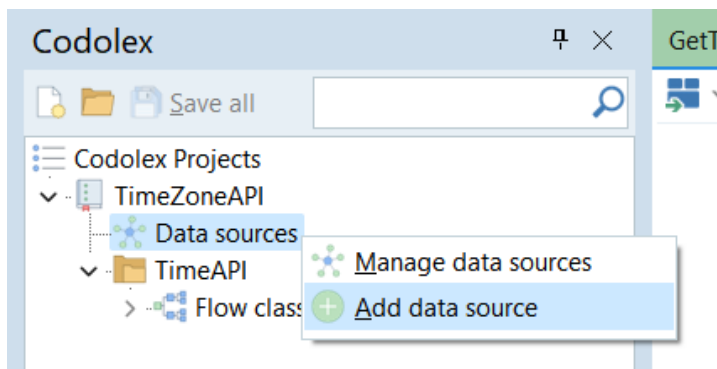
2.3.5 Adding a datasource

Previous topic: [Using flows in code](#) ¹⁶

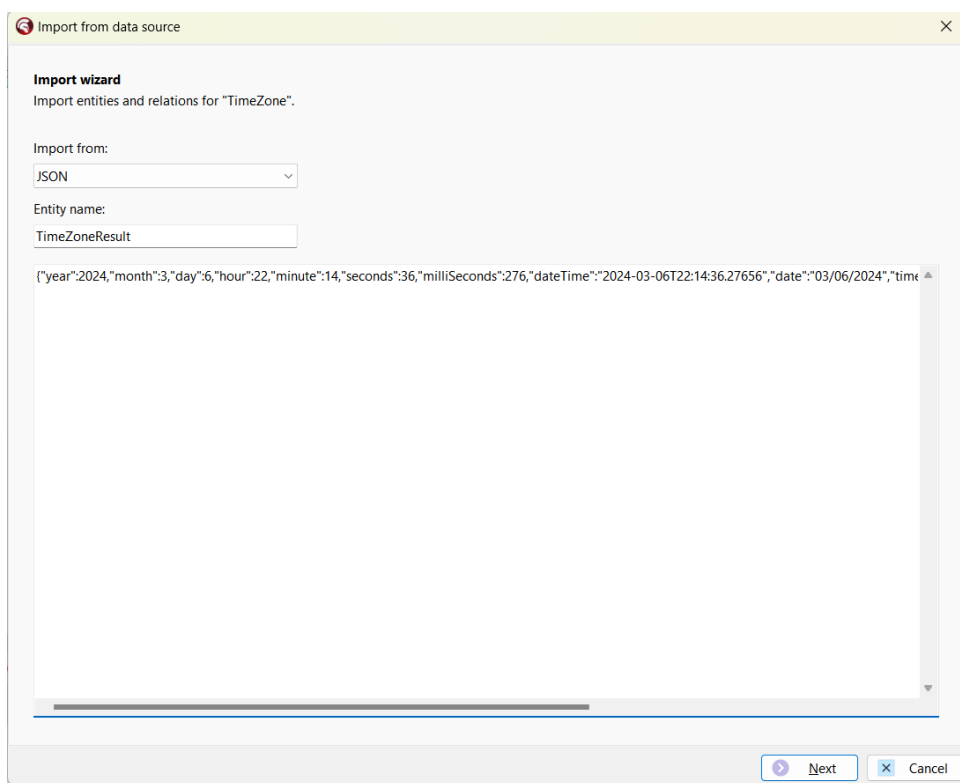
In the next steps, we are going to change the application to convert this JSON to a structured object. This is called "entities" in Codorex.

The first thing we have to do is to import the JSON structure. Luckily, Codorex has a handy tool for this (not only for JSON structures, you can also import directly from a database structure). Let's import the JSON structure in our example application.

To do so, right click on Data sources, and click on Add data source.



This will start the "Add data source" wizard. Give the Data source a name, for example, TimeZone, and choose REST as the data source type. Leave the "Import data" option to "Yes". After you click the save button, Codollex will open the "Import" wizard. Select JSON as the "Import" from option, give a name for the new entity that Codollex will create, for example TimeZoneResult. Now open the URL <https://timeapi.io/api/Time/current/zone?timeZone=Europe/Amsterdam>, copy the JSON and paste this JSON in the form.



Click Next. In the following screen, make sure to select the TimeZoneResult checkbox, click Next again, and finish the wizard.

To see how we can use this entity, go to the next section: [Using entities](#) ¹⁸

2.3.6 Using entities

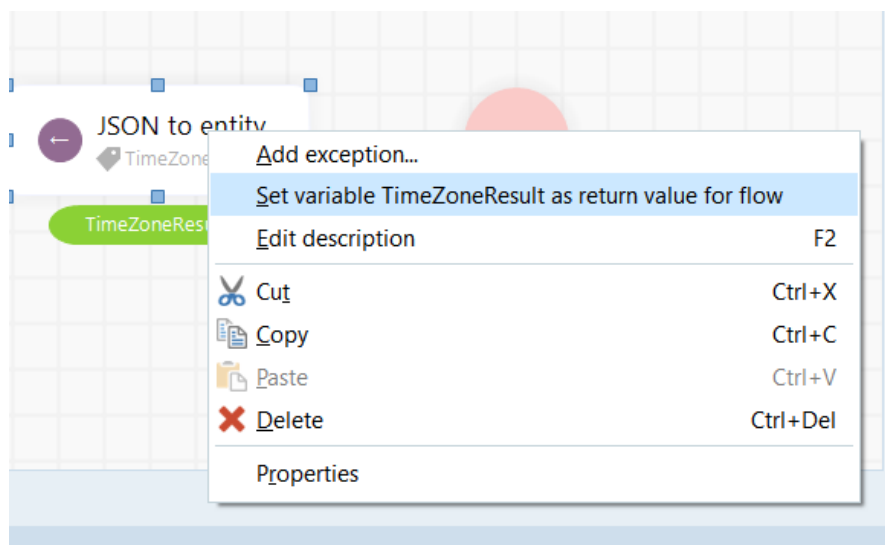
Previous topic: [Adding a datasource](#) ¹⁷

Codolux has now created a new entity with the structure based on the JSON result. Using entities makes it very easy to work with data in a structured way. Let us now modify the example application so that the result of the REST Operation is loaded into a new entity of the TimeZoneResult type.

First, go back to the GetTimeZone flow, and delete the ShowDialog activity. The REST Operation will return the JSONResult variable that we'll use to convert this variable to an entity. To do so, drag and drop a JSON to Entity activity in the flow, after the REST Operation, and double click on the JSON to Entity activity.

To convert the JSON string to an actual entity, click on the three dots next to the entity input, and select the TimeZoneResult entity. This is the structure we are going to import our JSON in. Next, select the JSONResult variable. This should be the only one available. You can keep the name of the result (TimeZoneResult) and click save.

Now, let's now change the flow to a function instead of a procedure so that we can give back the newly created entity. For this, right click on the JSON to Entity activity, and click on the option "Set variable TimeZoneResult as return value for flow".



You will see now that flow's endpoint displays the flow's result, which, in our case is an entity. To use this entity in code, let's change the OnClick code so we store the result of the flow. As an example, we can display one of the fields of our entity:

```
procedure TfrmMain.Button1Click(Sender: TObject);
begin
    var TimeZone: ITimeZone := TTimeZone.Create;
    var TimeZoneResult := TimeZone.GetTimeForTimeZone();

    ShowMessage(TimeZoneResult.dateTime.ValueOrDefault);
end;
```

By default, all fields in a Codolex entity are nullable. There are several functions available to work with these nullable fields, which will be explained in more detail [here](#) ⁵⁰.

This concludes the basic explanation of creating a Codolex application. In the next sections we will dive into the details of a Codolex project.

2.4 Codolex Concepts

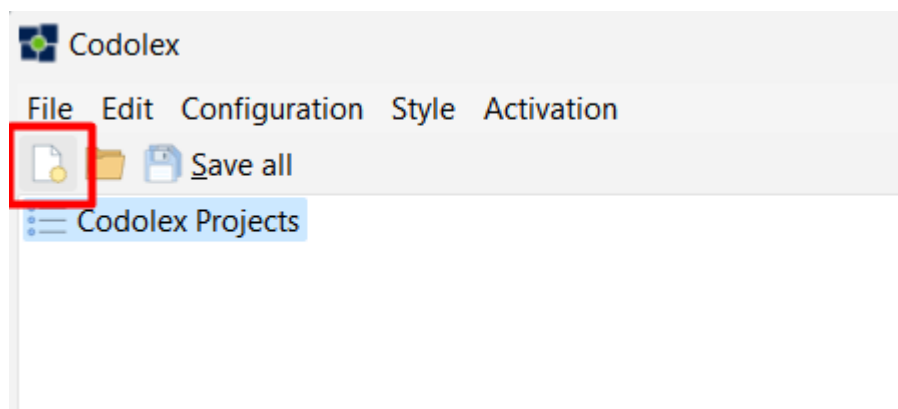
Codolex uses a project structure to store flows in a structured way. The structure is designed to be comprehensible for both small and large projects. With multiple layers for modules, classes, and flow classes, it's easy to separate logic and structure your application. In addition, the separation of data sources and local data sources allows for easy collaboration with different team members on a project.

All layers of Codolex will be covered in concepts for a quick understanding of a Codolex project.

2.4.1 Project explorer

The project explorer is where Codolex's project structure is visible. Several Codolex projects can be loaded within the explorer. This can be useful if you have multiple Codolex projects shared across multiple projects.

Using a right mouse click on "Codolex Projects", a new project can be created or loaded. To start a new Codolex project, click the "new project" button in the project explorer form:

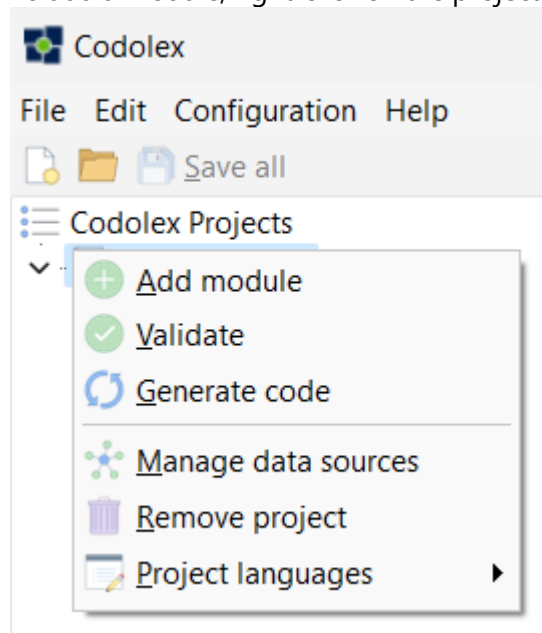


You will be prompted to save the project. Codolex will keep all info in subfolders of the folder containing the project.

2.4.2 Modules

A module is the highest level of the structure of a Codolex project. It is similar to a folder in the Delphi project structure, under which various source files are collected.

To add a module, right-click on the project and choose "Add module".

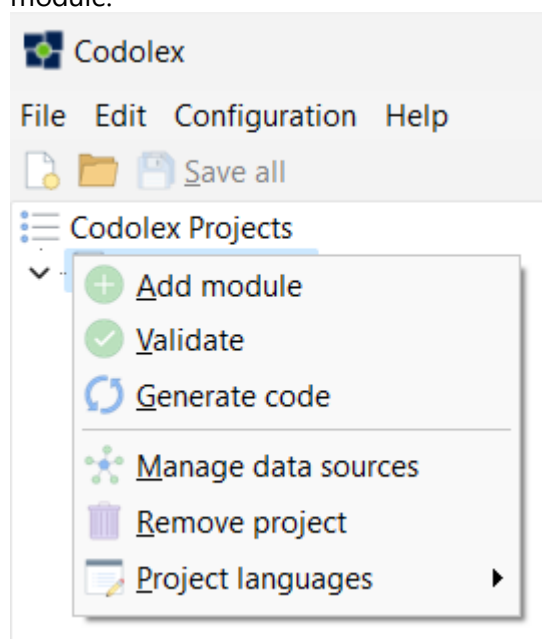


Enter a name for the module and click "Save". A module with this name will now be visible in the project explorer.

2.4.3 Flow classes

Codolex flow classes have the same purpose and meaning as classes in code. Within a class, you can define private, protected, and public methods, called flows.

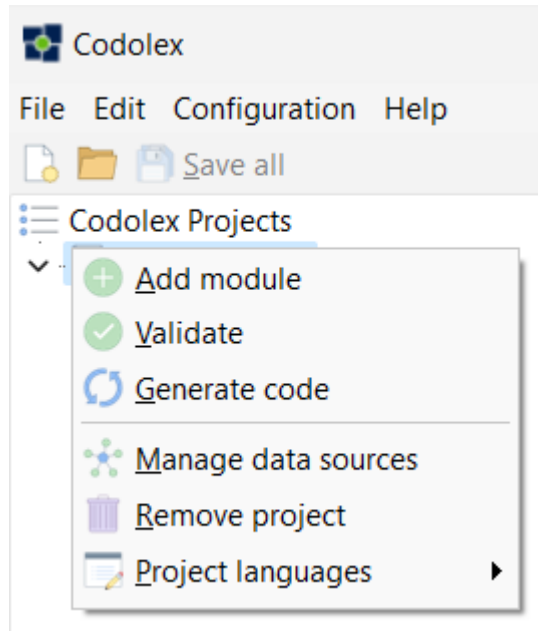
Create a flow class by right clicking on the "Flow classes" item below an open module.



Give a name for the flow class and click "Save". The flow class will appear in the project explorer. It is not necessary to use the prefix "T" for a class. Codolex will add the "T" when the source code is generated.

2.4.4 Flows

The flow is the equivalent of a procedure or function in a class. It's the lowest level of the Codorex structure, and it's here where you will define the business logic of an application. Add a flow by right clicking on your flow class in the project explorer. Choose "Add flow".



In the next form, choose the name and visibility for your flow and click "Save".

The flow will open in the editor. You are now ready to design code. Drag and drop activities from the palette to create business logic for your application.

2.4.5 Data sources

A data source is a collection of entities. These can be database entities (tables or views), in-memory entities or REST entities. This data can be accessed through a data source. When distributing applications, you need to load the data through an actual connection. See [Connecting to databases](#)^[48] for more information about use data sources.

2.4.6 Local data sources

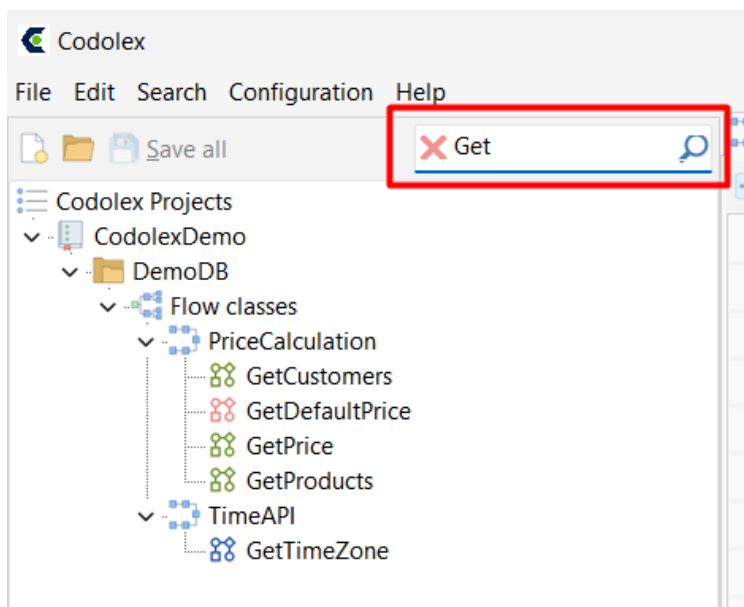
A local data source is a connection to a database on a development machine. This provides a way to import data structures, and test the database connectivity.

2.4.7 Entities

In Codorex, entities are representations of data. This can be data from a database, but also from residual services or other sources. See [Entities](#)^[49] for more information about entities.

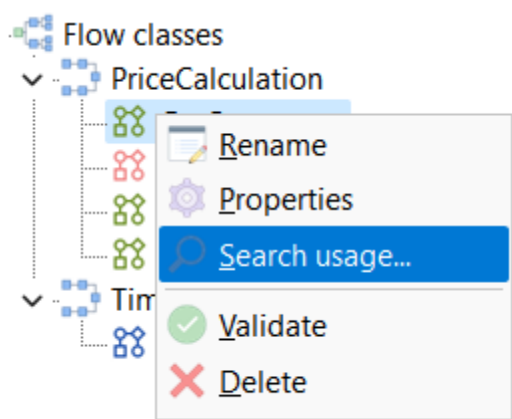
2.4.8 Search

When you have a large Codolux project open, there are several ways to search and navigate through the project. First is the filter option in the program explorer window:





This makes it easy to filter flow classes and flows by name.

In addition, you can see in several ways whether a flow, an entity or a data source are used somewhere in Codolux. To do this, you can right-click on one of the relevant elements and choose "search usage":



A screen will now open showing all relevant places. By double-clicking, you can open the relevant flow and activity.

Entity usage: "PriceCalculation.CustomerPrice"				
Document	Element	Name	Property	Text
>  CodolexDemo / CodolexDemo / PriceCalculation	Association	CustomerPrice_Custom	FromEntity	CustomerPrice
 CodolexDemo / DemoDB / PriceCalculation / Gi	Activity		ReturnVariable.Entity	CustomerPrice

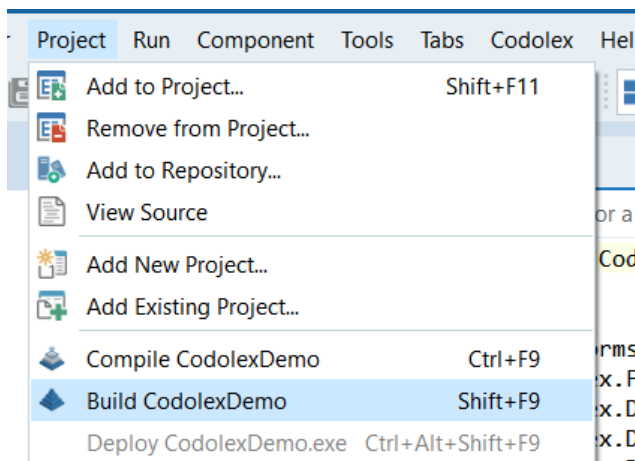
Code generation

3 Code generation

With Codolex, you design business logic via flows. However, these flows always result in source code that needs to be compiled. Codolex can generate this code automatically, but it is also possible to create the code manually.

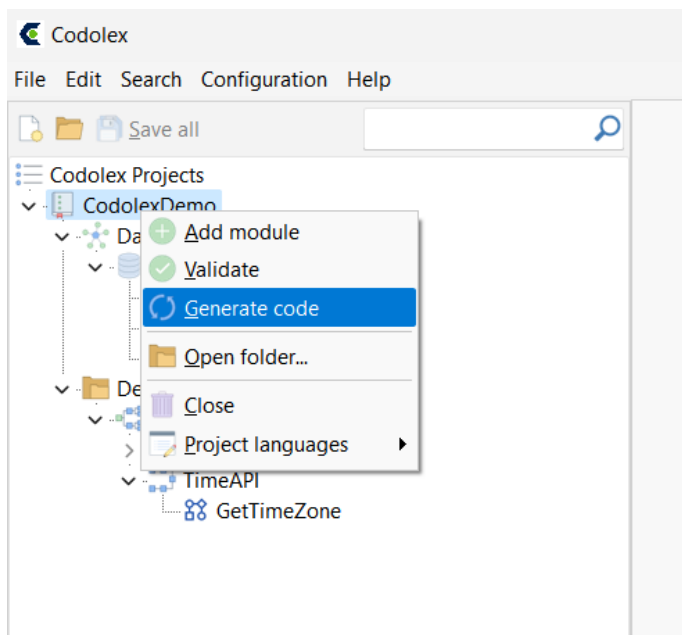
3.1 Automatic code generation

Once a Codolex project is linked to your Delphi project, the Delphi plugin of Codolex will automatically generate the source code the moment you compile or build the Delphi project. The generated source code will be stored in a new subfolder named `.fsrc`. You can choose to add this folder to your code repository too, but do not make any manual changes to the generated files. Your changes will be overwritten if you compile your Delphi project.



3.2 Manual code generation

It is possible to generate the source code using the standalone app of Codolex, or using the [command line interface](#)^[220]. To generate code using the Codolex standalone app, right click on your Codolex project file, and click "Generate code".



3.3 File structure

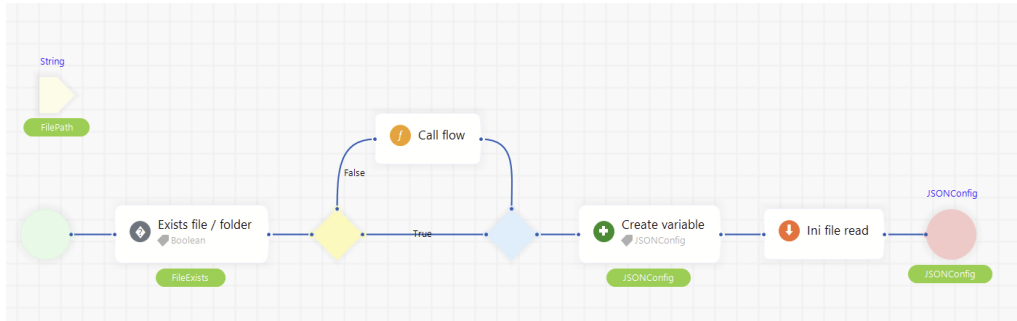
Codolex projects have the file extension `.fcp`. In the location where you save the Codolex project file, a new folder with the prefix `.fc` will be created. This folder contains all the module and flow files. Add these files to your code repository.

The generated source code will be stored in a new folder named `.fsrc`. You can choose to add this folder to your code repository too, but do not make any manual changes to the generated files. Your changes will be overwritten if you compile your Delphi project.

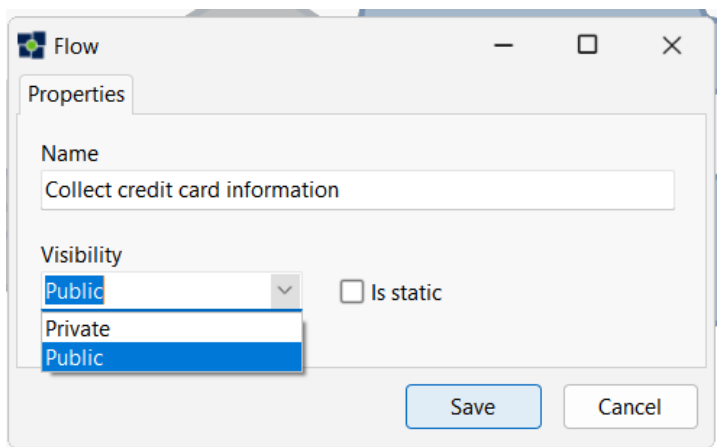
Flows

4 Flows

A flow is the basis on which Codolex's business logic works. A flow consists of several actions or activities, ultimately resulting in source code.

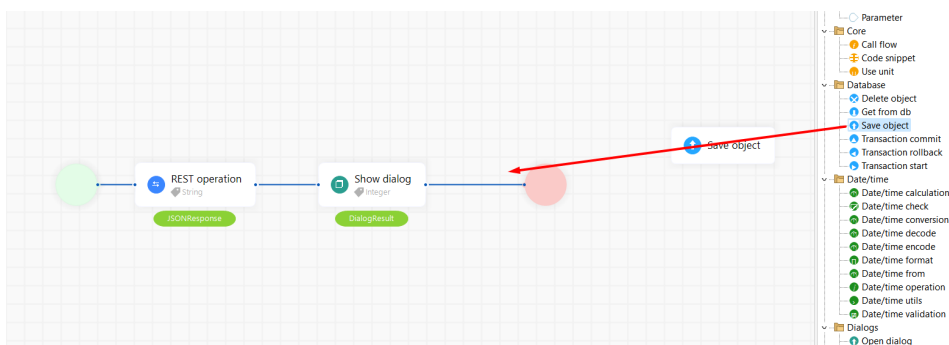


A new flow can be created via the project explorer. This can be done by right-clicking on a flow class, and choosing "Add flow". A screen follows to set some parameters of the new flow.

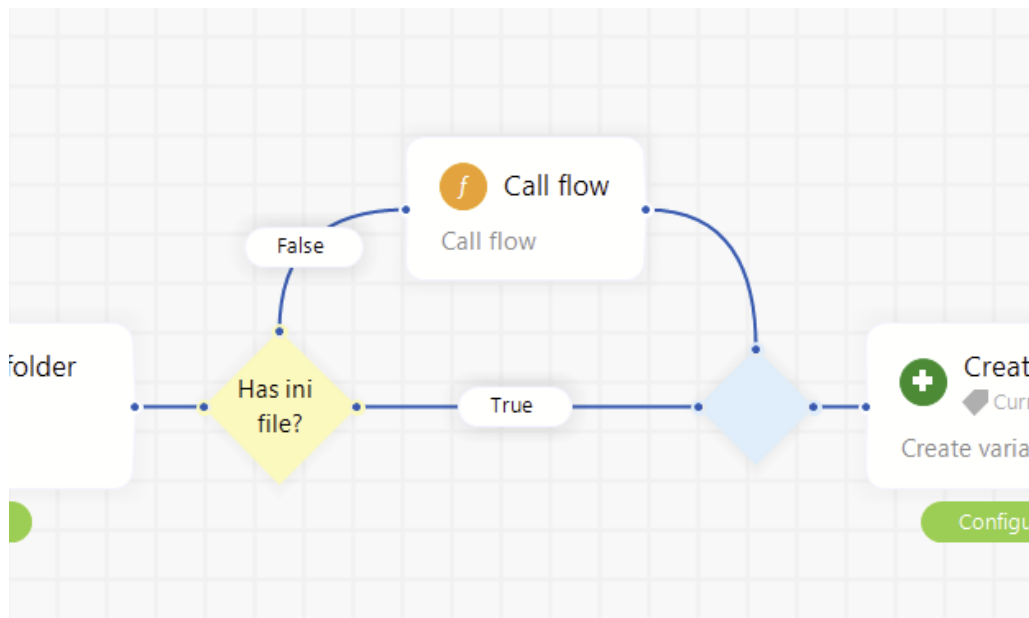


4.1 Flow handling

A flow consist of activities and sequences. The activities determine the generated logic, and the sequences determine the relationship between the activities. To add an activity to a flow, simply drag and drop an activity from the activity palette to the flow.



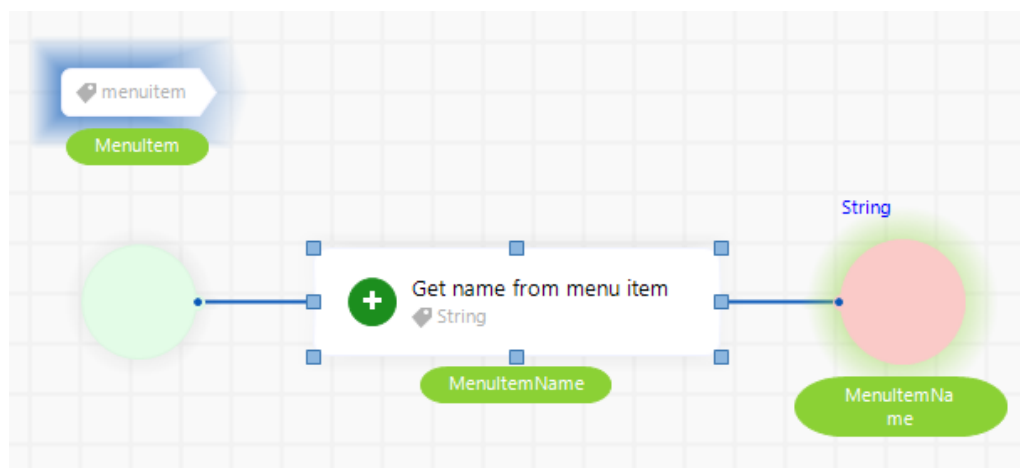
Sequences connect individual activities. Sequences allow you to set the order of a flow. An activity always needs one or more incoming sequences and can have one or more outgoing sequences.



Look for the green and blue highlights to see the usage of flow variables.

Green highlight means, the return variable or parameter from the selected activity is used here.

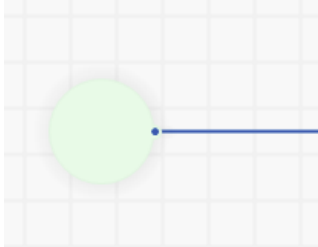
Blue highlight means, the return variable or parameter is used in the selected activity



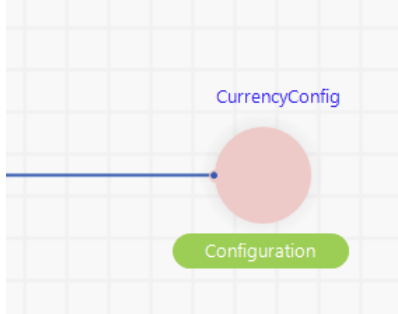
Variable usage example

4.2 Start and End

Every flow has exactly one Start activity. This activity is created automatically and can't be deleted from a flow. There is one sequence from the start activity to the first activity of a flow.

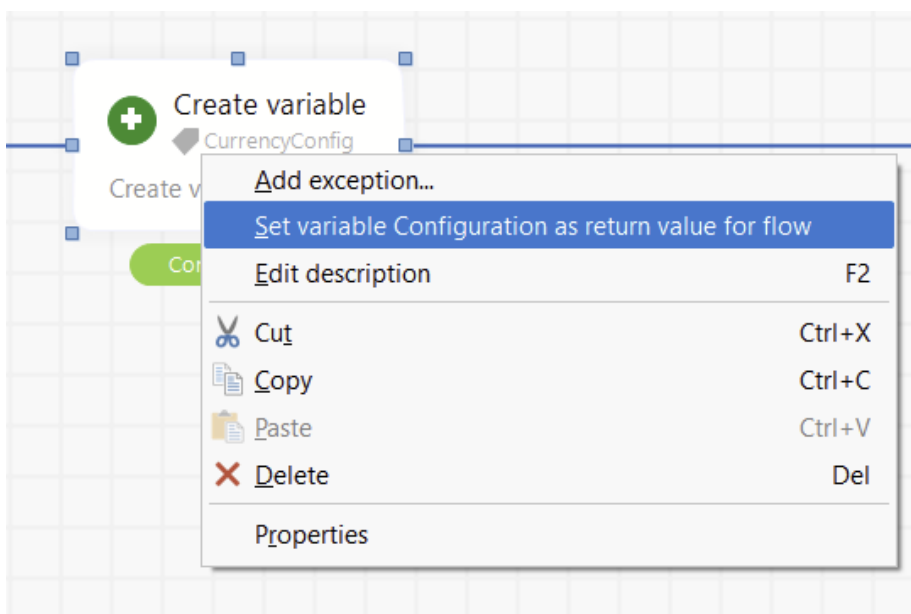


A flow can have multiple End activities. The End activity will stop the execution of the flow and can have a result value to set the result of the flow. Every End activity will have the same result type (e.g. Boolean, Integer, string or custom type).



4.3 Flow results

To set the flow result, double-click on the End activity, or right click and select "Set variable as return value for flow" on an activity.

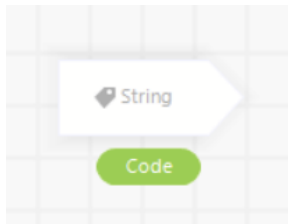


If you set a flow result, the Codollex will generate the flow code with the corresponding result:

```
procedure FlowWithoutResult;
function FlowWithResult: TCustomerEntity;
```

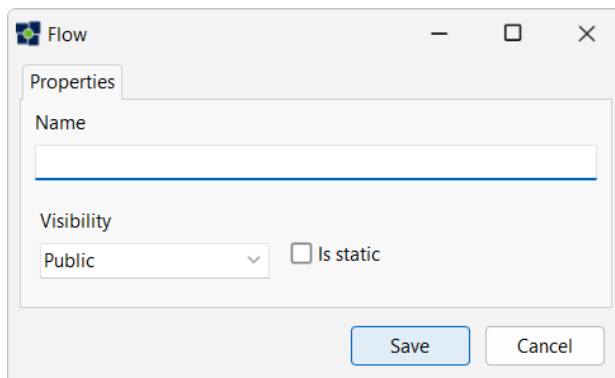
4.4 Parameters

Variables can be passed to flows via parameters.



More info: [Parameters](#) ⁵⁷

4.5 Visibility



A flow can be private and public. A private flow means that this flow can only be called within the flow class where the flow is located. A public flow can also be called from other flow classes. This way, the visibility of a flow can be controlled.

A static flow means that the flow can also be called independently, without creating a flow class with the generated Delphi code. In the example below, this flow is invoked as static:

```
TestProject.Suppliers.Invoices.DeleteCustomer(Customer);
```

If the flow is not static, the flow class must be created as well:

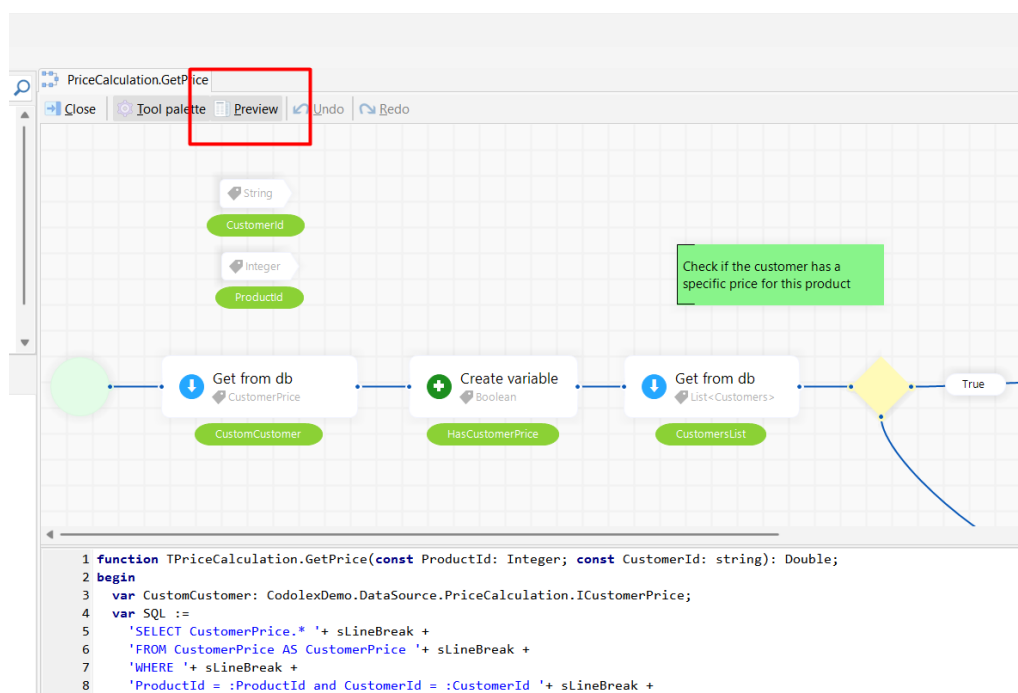
```
var Invoices := TestProject.Suppliers.Invoices.TInvoices.Create;
try
  Invoices.DeleteCustomer(Customer);
finally
```

```
Invoices.Free;  
end;
```

Use static flows for business logic that does not depend on the state of the flow class. For example; if you have a flow to send messages to customers, where the customers are collected and stored in a flow class variable, don't use the static flag.

4.6 Code preview

To preview the code that Codolox will generate, click the preview button in the menu:

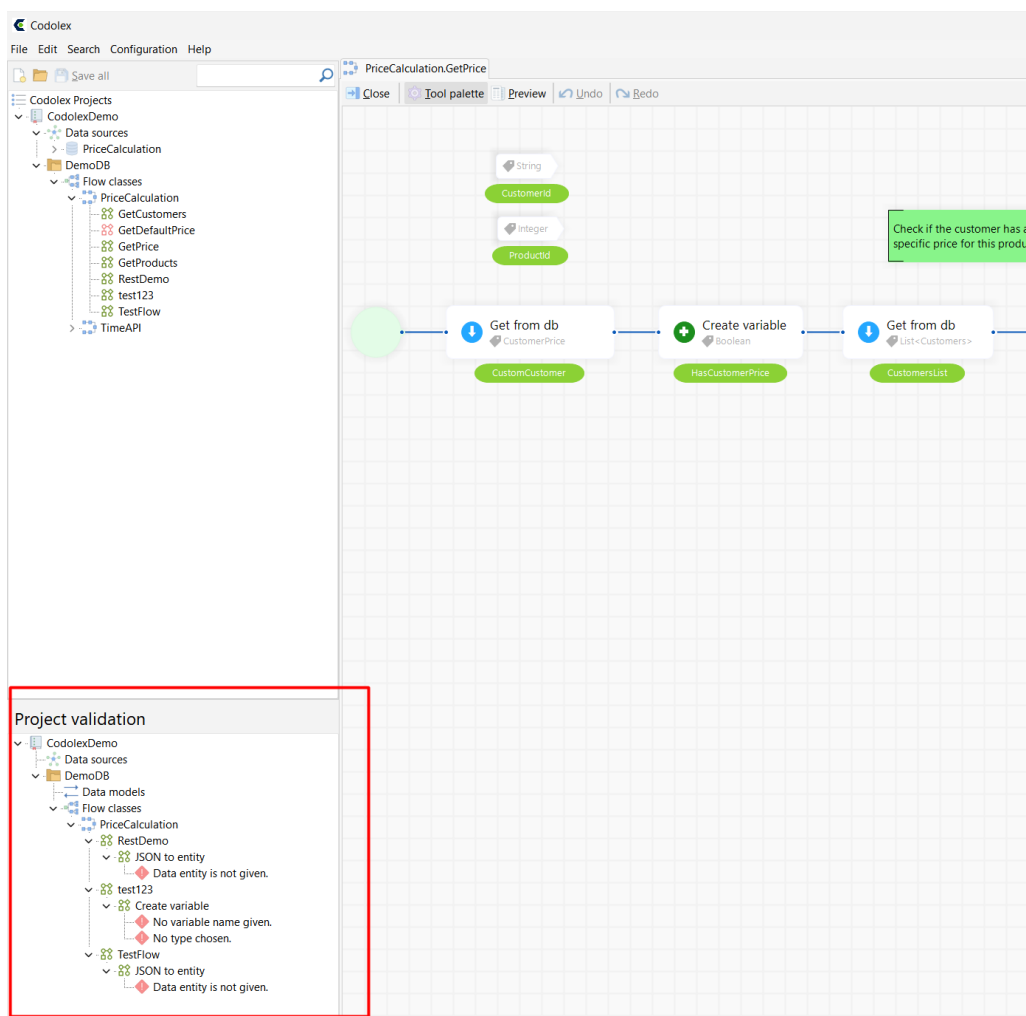


You can leave the preview windows open to see the code changes immediately when updating flows.

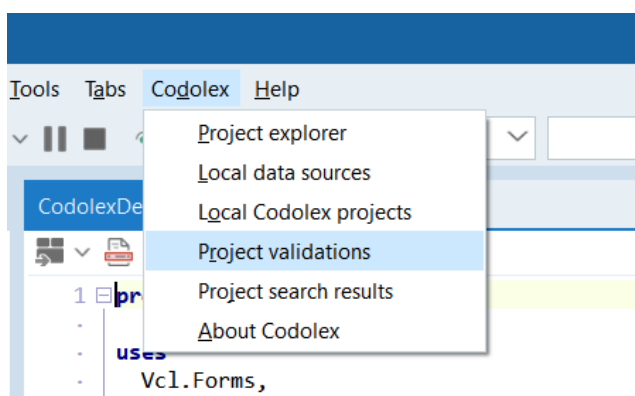
It's also possible to see the code preview for a complete flow class. To see this, right click on a flow class, and click "Preview source code".

4.7 Validation

Codolox has a comprehensive validation system, to help you find errors, warnings and hints. If you use the standalone version of Codolox, you always see the project validation in the main screen:



When using the Codolex integration of Delphi, you can open the project validation results from the menu:



By double-clicking on the error, the corresponding flow will open and the activity will be selected.

Codolex API

5 Codolex API

It's possible to expose an API with codolex.

This could be useful for making a coupling with another application, retrieving web-hook triggers, exposing data, etc.

The API can be configured to offer certain endpoints through a flow, or direct database access through a data source.

The following topics will help in setting up and configuring an API

[Setup API](#) 

[Exposing Data](#) 

[Adding endpoints with flows](#) 

[Configure security](#) 

5.1 API Setup

Structure

Setting up an API with Codolex is very simple.

By default the generated code includes a folder

"**\\fsrc\\[yourcodolexprojectname]\\API**". this folder contains the following files as basis.

- [yourcodolexprojectname].API.Authentication.pas
- [yourcodolexprojectname].API.Controller.base.pas
- [yourcodolexprojectname].API.Server.pas
- [yourcodolexprojectname].API.Server.dfm

If you have a Codolex project linked, the generated folder is included in the search path by default.

The ...Server.pas file contains everything you need to start the API.

```
TCodolexAPIProjectApiServer = class
private
    FWebBrokerBridge: TIdHTTPWebBrokerBridge;
public
    procedure AfterConstruction; override;
    procedure BeforeDestruction; override;

    procedure OnAuthentication(const Action: TOnAuthenticationEvent);
    procedure WithSwaggerInfo(const UseSwaggerInfo: TUseSwaggerInfo);

    procedure Start(const Port: Integer);
    procedure Stop;

    function IsActive: Boolean;

    property ServerInstance: TIdHTTPWebBrokerBridge read FWebBrokerBridge;
end;
```

You can include this file anywhere in your project. and start the API with the following code

```
begin
  var APIServer := TCodoloxAPIProjectApiServer.Create;

  APIServer.Start(8080);
end;
```

Configuration

The server is normally run on the localhost without a certificate. The Sever instance is available as property to configure the server. This can be used set a certificate for example.

```
var ServerInstance := APIServer.ServerInstance;
var Handler := TIdServerIOHandlerSSLOpenSSL.Create;

Handler.SSLOptions.Mode := sslmBoth;
Handler.SSLOptions.KeyFile := KeyFileLocation;
Handler.SSLOptions.CertFile := CertFileLocation;
Handler.SSLOptions.RootCertFile := RootCertFileLocation;
Handler.SSLOptions.SSLVersions := [sslvTLsv1, sslvTLsv1_1,
sslvTLsv1_2];

ServerInstance.OnQuerySSLPort := OnQuerySSL;
ServerInstance.IOHandler := Handler;
```

Next to the server instance, there are 2 public functions that can be used for configuration.

1. OnAuthentication

The procedure expects an action to validate the user. The 'TOnAuthenticationEvent' type is the following function

```
TOnAuthenticationEvent = reference to function(const UserName,
Password: string; const UserRoles: TList<string>; const
SessionData: TDictionary<string, string>): Boolean;
```

This function gets called before every request to check if the user has the right access to the object/flow.

The result must be a boolean. you can check the *UserName* and *Password* that the user provides to authorize the request.

The roles will be further explained in the section [Configure security](#) ⁴³

2. WithSwaggerInfo

The API generates Swaggerinfo by default. This swagger info can be given extra details with the 'TUseSwaggerInfo' function.

```
TUseSwaggerInfo = reference to function: TMVCSwaggerInfo;
```

The function has the return the TMVCSwaggerInfo object. This object has some attributes that can set the swagger information.

```
var Title: string;
var Version: string;
var TermsOfService: string;
var Description: string;
var ContactName: string;
var ContactEmail: string;
var ContactUrl: string;
var LicenseName: string;
var LicenseUrl: string;
```

So the swagger info can be provided like this:

```
var SwaggerInfo: TUseSwaggerInfo := function: TMVCSwaggerInfo
begin
    Result.Title := 'TestProject API Server';
    Result.Version := '2.4.0';
    Result.Description := 'TestProject API Server generated by
Codolex';
    Result.ContactEmail := 'info@codolex.com';
end;

var APIServer := TCodolexAPIProjectApiServer.create;
APIServer.WithSwaggerInfo(SwaggerInfo);
```

Stop server

The server can be stopped by calling the **Stop** function, or by closing the application.

5.2 Exposing Data

Available data

When the API is configured and started, the database datasources from the project are exposed as data to retrieve and set.

For every entity in the database, the following calls are available:

```
Post -> ServerURL/[DatasourceName]/[Entityname]/
Patch -> ServerURL/[DatasourceName]/[Entityname]/{Identifier}
Delete -> ServerURL/[DatasourceName]/[Entityname]/{Identifier}
Put -> ServerURL/[DatasourceName]/[Entityname]/{Identifier}
Get -> ServerURL/[DatasourceName]/[Entityname]/{Identifier}
```

Example:

A server hosted on localhost with the project containing a datasource "CodolexData" with a table "Orders", you can retrieve order with ID 80 by

"<http://localhost:8080/CodorexData/Orders/80>"

Swagger documentation

The server contains an url to retrieve swagger data in JSON for the API.

So the documentation can be always be found at

"**[ServerURL]/api/swagger.json**" by default.

Use a website like <https://editor-next.swagger.io/> to view the documentation in a structured manner.

DMVCFramework System Controller

When viewing the swagger documentation, one other base for operations can be found, the "DMVC Framework System Controller"

These calls can be used to get specific information about the server and platform.

Severconfig -> information about the server, like request size and timeout

DescribePlatform -> information about the host of the server

DescribeServer -> information about the available endpoints in simple JSON format. This includes the database entitties and flow.

5.3 Adding endpoints with flows

Every public flow in a project has the option to become an endpoint for the API.

To expose a flow, you need to set the property **Publish as endpoint for API**.

The screenshot shows a 'Properties' dialog box for a flow named 'HelloWorld'. The 'Name' field is 'HelloWorld'. The 'Visibility' dropdown is set to 'Public', and the 'Is static' checkbox is unchecked. The 'Publish as endpoint for API' checkbox is checked. The 'Endpoint' tab is selected, showing the 'REST/HTTP method' dropdown set to 'Automatic' and an empty 'Custom path' field. The 'Flow' dropdown is set to 'HeaderFlow'. At the bottom, there are 'Save' and 'Cancel' buttons.

There are a few options in the configuration that need some explanation.

Endpoint

REST/HTTP method

This defines the type the method is going to get. By default this is automatic. This means that the method will always be **GET**, except if the flow contains a parameter of type entity. That means that the parameters is expected as body, so a **POST** is needed to retrieve a body.

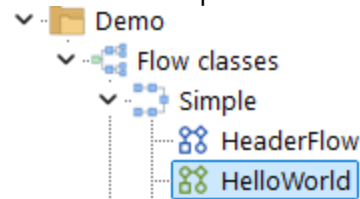
The other available methods are

- GET
- POST
- PUT
- PATCH
- DELETE
- HEAD
- OPTIONS
- TRACE

Custom path

By default the path to call this method is defined by the module, flow class and name of the flow.

So in this example



The path to call 'Hello world' would be: "*ServerURL/demo/simple/helloworld*".

When using the custom path, the name is replaced. So if we would take the example again, and set the custom path to 'directpath', our URL to call would look like: "*ServerURL/demo/simple/directpath*".

To make multiple paths available for the same flow, use a semi colon in the custom path field between the paths.

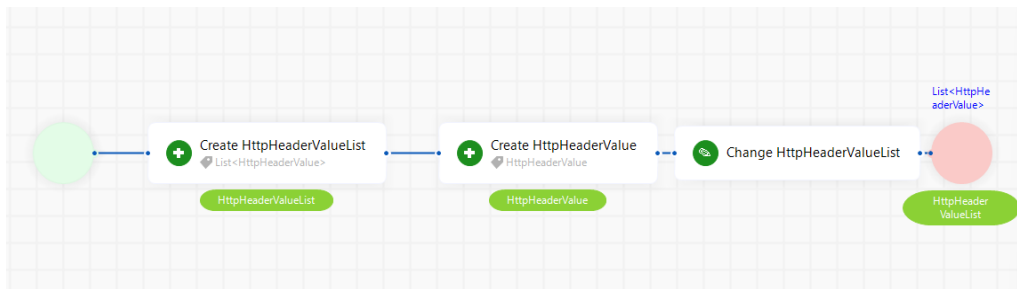
(Header) Flow

The flow option can be set to a flow that provides header values for the response.

This flow must be a **static** flow with the result set to **list of HttpHeaders**.

(Can be found in Core.Activities.DataSource)

The header value must contain a 'key' and 'value' string.



Example flow

Authorization

On the authorization tab, you can define what roles have access to this flow.

The screenshot shows the 'Authorization' tab of the Codorex API interface. It features a checkbox labeled 'Is anonymously accessible' which is checked. Below this, there is a section titled 'Roles with access' containing a table with a single header 'Name' and an empty body.

The other option is to set the flow as **Anonymously accessible**, in this case, no login is needed to call the flow.

More information about logging in and roles can be found at [Configure security](#)⁴³

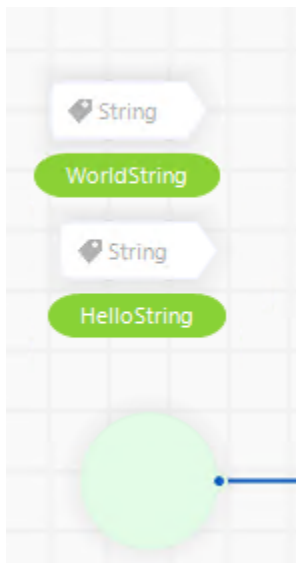
Parameters and return value

The structure of a flow changes the way the endpoint works. while parameters are usefull for retrieving data, the result can be set to text or an entity to send data to the caller.

Parameters

Ordinal types of pramaters can be used, this changes the url into 'Module/Flowclass/Flowname/**paramater1/parameter2**'

take these paramters as example for the hello world flow mentioned above:



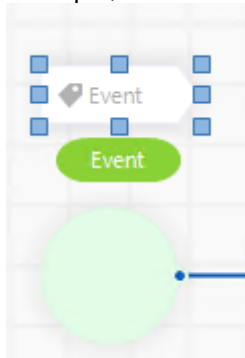
Calling "ServerURL/demo/simple/helloworld/world/hello" would fill the 'WorldString' and 'HelloString' parameters with 'hello' and 'world' depending on the order of the parameters.

Entities can also be used as parameters, these will not be included in the url, but in the body.

Create a JSON of the entity and provide it as a string in the body, and use a parameter for the entities.

When providing multiple entities, use a list.

Example, when using event as a parameter:



provide this json in the body:

```
{
  "EventID": 1,
  "EventDate": "11-11-2024",
  ....
}
```

The **HttpRequest** entity can also be used as a special parameter, this will be used next to the existing parameters and provide information about the request.

it contains information like the caller host or content type, and has associations to:

- Header values
- Query values
- Form values

The full structure of this entity can be found in the plugin datasource

Core.Activities.Datasource.

Return value

The return value can be a string or an entity, the string will be returned as raw string, and the entity as a JSON.

Keep in mind that associations are parsed with the entity, so if 'event' has an association to 'location', the return value on the request can look something like this:

```
{
  "EventID": 1,
  "EventDate": "11-11-2024",
  "Location": {
    "LocationID": 2,
    "LocationName": "The office",
    ...
  }
  ....
}
```

5.4 Configure security

When setting or getting data with the API, the call must be authenticated with user name and password.

The only exception on this are the information flows from the framework, and the flows that are anonymously accessible.

User name and password

To check if the right user name and password are provided, the Authenticate flow must be provided.

```
TOnAuthenticationEvent = reference to function(const UserName,
Password: string; const UserRoles: TList<string>; const
SessionData: TDictionary<string, string>): Boolean;
```

Include in uses: System.Generics.Collections

This is a property that can be set on the API.

```
var APIServer := TCodollexAPIProjectAPIServer.create;
```

```
APIServer.OnAuthentication(AuthenticateUser);
```

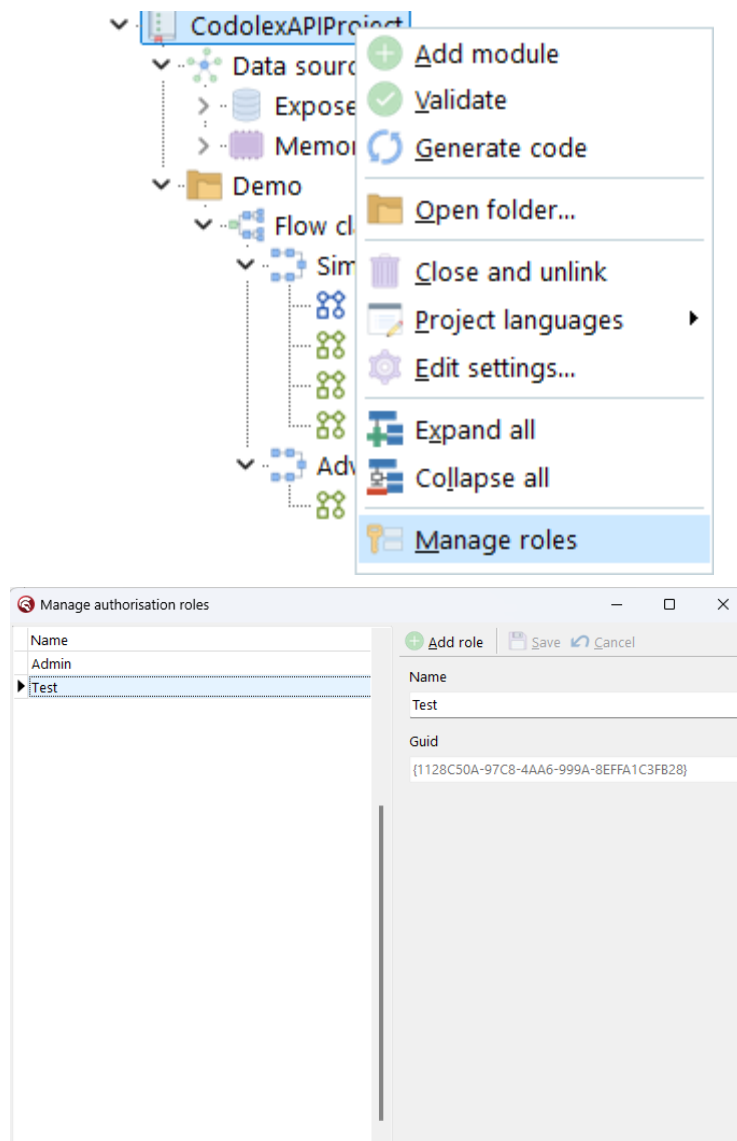
assuming the *AuthenticateUser* flow is a function of the given type.

In this function, the given user name and password are given as parameters, In the flow, you can check against a specific value or database if the data is valid. Set the result value to **True** when the data is valid. otherwise the response will always be 403 - Unauthorized

Roles

In addition to username and password, you can also limit access to flows through roles.

These roles must be configured in the project



In the flow properties, you can set 1 or more of these project roles.

When using the Authenticate flow, add roles to the **UserRoles** list to specify the roles that belong the user with the given user name and password.
If one Role is present in this list and in the defined roles for a flow. the user is able to call te flow through the api.

Example

The following code is an example of how the authentication function can be defined.

```
function TForm1.AuthenticateUser(const UserName, Password: string;  
const UserRoles: TList<string>; const SessionData:  
TDictionary<string, string>): Boolean;  
begin  
    Result := False;  
  
    if (username = 'test') and (password = 'test') then  
    begin  
        UserRoles.Add('Test');  
        Result := True;  
    end;  
  
    if (username = 'admin') and (password = 'admin') then  
    begin  
        UserRoles.Add('Test');  
        UserRoles.Add('Admin');  
        Result := True;  
    end;  
end;
```

*Note that harcoded users are used for example purposes. This is **not** our recommendation for a exposed API.*

Data sources

6 Data sources

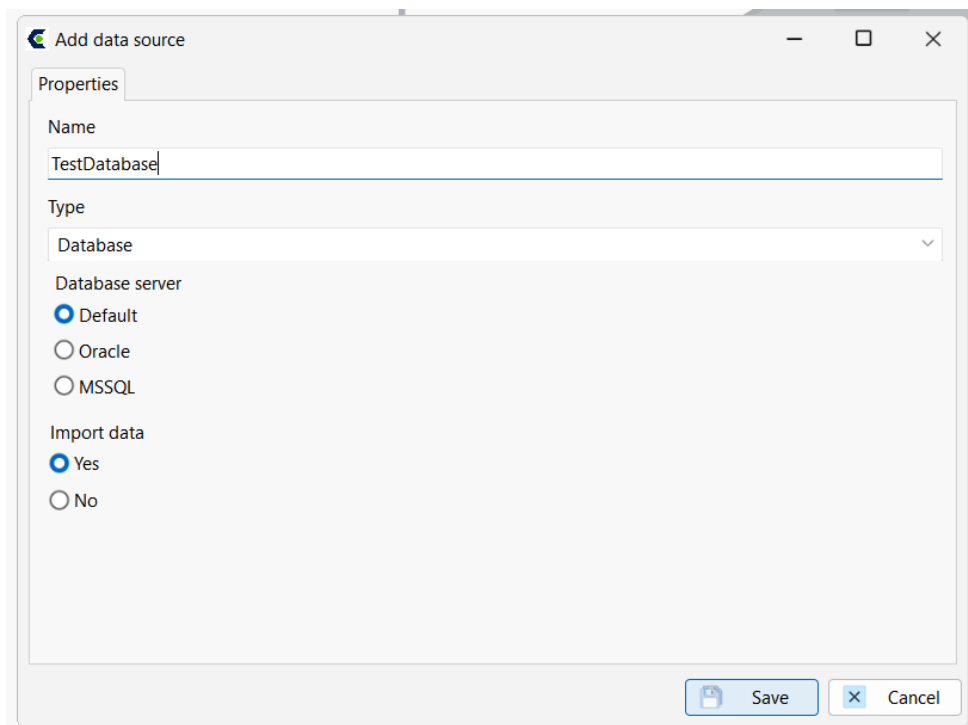
A data source is a collection of entities. These can be database entities (tables or views), in-memory entities or REST entities. This data can be accessed through a data source.

6.1 Local data sources

A local data source is a connection to a database on a development machine. This provides a way to import data structures, and test the database connectivity.

To get an overview of all local data sources, choose "Configuration" (in the standalone app) or "Codolux" (In Delphi) and "Local data sources" from the menu. This overview shows all local data sources created for the current system. These local data sources are a local link to the data, and so are not directly linked to a Codolux project itself.

You can also create a local data source via the add data source wizard. To do this, right-click on Data sources, choose "Add data source" and under Import Data, choose "Yes".



In the subsequent screen, enter the database details of the local data source. Click on "Test connection" to see if the database connection has been created properly. Then click "Next" to import any data.

6.2 Connecting to databases

To use database activities such as "Get from DB", Codolex must be able to connect to a database. This requires the Delphi project that Codolex uses to have at least the data sources properties configured. The easiest way to retrieve the connection code is to right-click on a data source, and click "Copy connection code". Codolex will place the connection code in the clipboard, so you can place this in the initialization section (for example in the DPR file or in a data module) of your application.

To add database access manually, include the following units to your DPR file or to the initialisation code of your project:

```
Codolex.Framework,  
Codolex.Database.Query.Interfaces,  
Codolex.Database.Query.FireDAC,  
Codolex.Database.Connection.FireDAC,  
System.SysUtils,
```

Define a function that provides an implementation of `IDatabaseQuery`. This interface allows Codolex to communicate with the database. There is a default implementation available for FireDAC. Configure this function, the query provider, as follows:

```
var QueryProvider: TFunc<IDatabaseQuery>;  
// It assumes that there is a FireDAC connection available (MyConnection)  
QueryProvider :=  
  function: IDatabaseQuery  
  begin  
    var DbConnection := TDatabaseConnectionFireDAC.Create(MyConnection);  
    Result := TDatabaseQueryFireDAC.Create(DbConnection);  
  end;
```

The Codolex Framework is the bridge between your application and the generated sources. Use the framework to link the database query provider to your data source by name.

```
// The name of the data source is e.g. 'MainDatabase'  
CodolexFramework.DatabaseQueryProvider['MainDatabase'] := QueryProvider;
```

Add the following compiler directive to your Delphi project, for example in the DPR file:

```
{$STRONGLINKTYPES ON}
```

Codolex uses RTTI information to retrieve the Databroker for entities dynamically.

6.3 Custom database connections

See [this YouTube video](#) for more information

6.3.1 Nexus DB

Files needed:

- Codolex.Database.Connection.NexusDb.pas
- Codolex.Database.Query.NexusDb.pas
- Codolex.Database.Param.DataDb.pas

[download here](#)

Code for integration

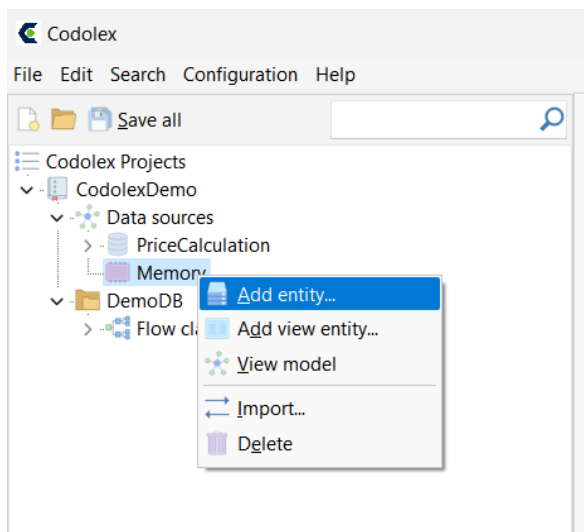
```
uses
  Codolex.Framework,
  Codolex.Database.Query.Interfaces,
  Codolex.Database.Connection.NexusDb,
  Codolex.Database.Query.NexusDb;

var NexusQueryProvider: TFunc<IDatabaseQuery>;
NexusQueryProvider := function: IDatabaseQuery
begin
  // Wrap the Local TnxDatabase component in the
  var Connection := TDatabaseConnectionNexusDb;
  // Create a query wrapper that uses the TnxQuery
  Result := TDatabaseQueryNexusDb.Create(Connection);
end;

// Link the provider function to your datasource from the Codolex module
// Replace MainDatabase with the name of your database datasource
CodolexFramework.DatabaseQueryProvider['MainDatabase'] := NexusQueryProvider;
```

6.4 Entities

In Codolex, entities are representations of data. This can be data from a database, but also from REST services or other sources. When importing a table structure from a database, entities will be created automatically, just like when importing a REST service. But it is also possible to create entities manually. To do this, right-click a data source, and click Add Entity or Add View Entity:



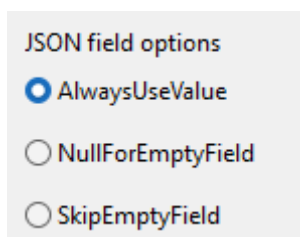
The difference between a normal entity and a view entity is the ability to modify: a view entity (like a view from a database) is readonly, and can consist of several other entities.

6.5 Nullable fields

If you use Codolex entities in regular Delphi code you will see that all entities in Codolex are of type `TCodolexField`, for example `TCodolexField<Integer>`. The following functions are available for a `TCodolexField` (in this example, a `TCodolexField<string>`

constructor	Create (const Value: string);
procedure	Clear ;
property	IsNullable : Boolean;
property	HasValue : Boolean;
property	IsNull : Boolean;
property	IsChanged : Boolean;
property	Value : string;
property	ValueOrDefault : string;
property	Empty : <code>TCodolexField<string></code> ;

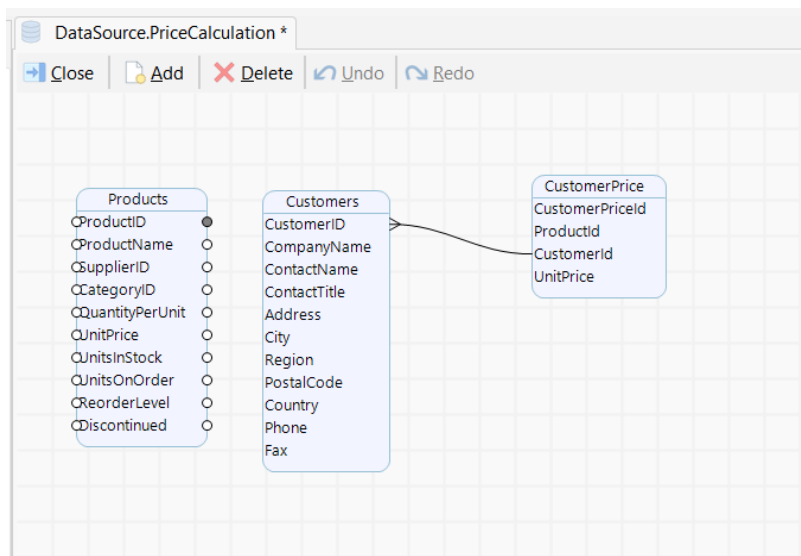
For exporting data, there is an option how null-able fields should be handled. It contains 3 options, null, 0 or '' values, or exclude from the JSON.



6.6 Importing data structures

To import a data structure, right-click on Data sources, choose "Add data source" and under Import Data, choose "Yes", or right-click on an existing data source and click Import. If you want to import data structures from a database, Codolex will list the existing tables and views. After importing, these are displayed as entities under the data source. If the data source is a different type, you can import data structures from JSON or CSV. To do this, paste the complete JSON or CSV into the wizard, and Codolex will create the correct entities based on it.

Codolex will also import the relationships between tables and JSON structures. To view these relationships, double-click on an entity and open the Associations tab. To view all the relationships in a certain data source, right-click on the data source and choose View Model. Codolex will open the data view including all the entities of the data source.



6.7 IniFile Datasource

To read values from ini files for settings or other data an ini file datasource can be used.

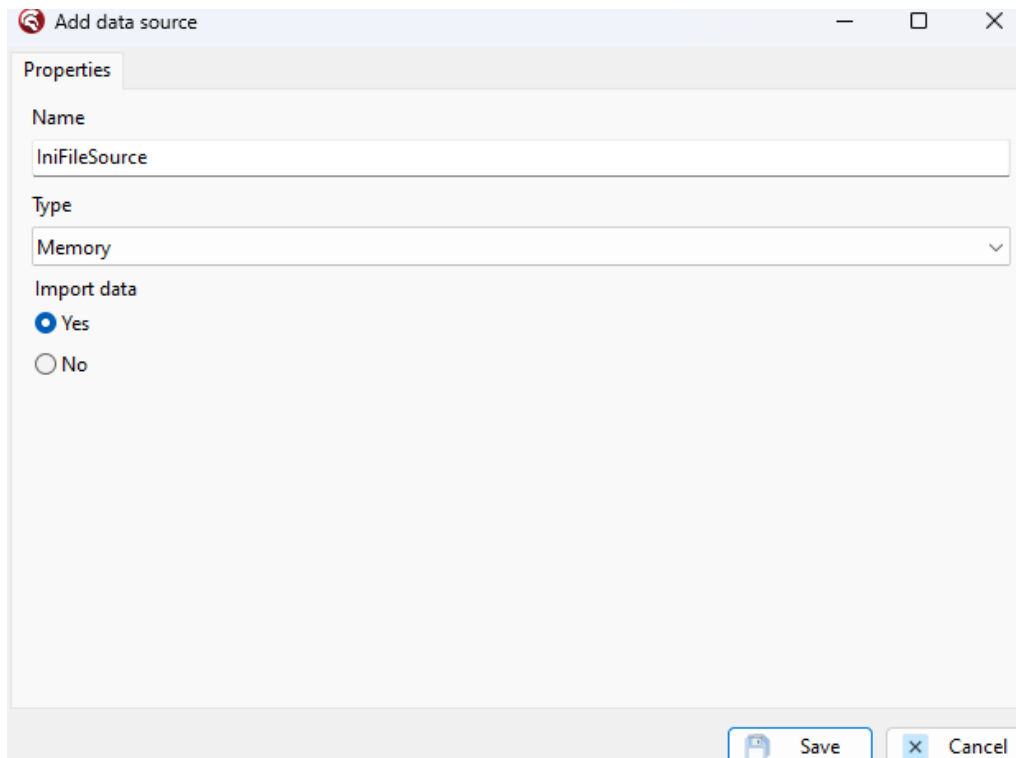
This datasource is a memory datasource that can be used in the read and write activity

[IniFile Read](#) 130

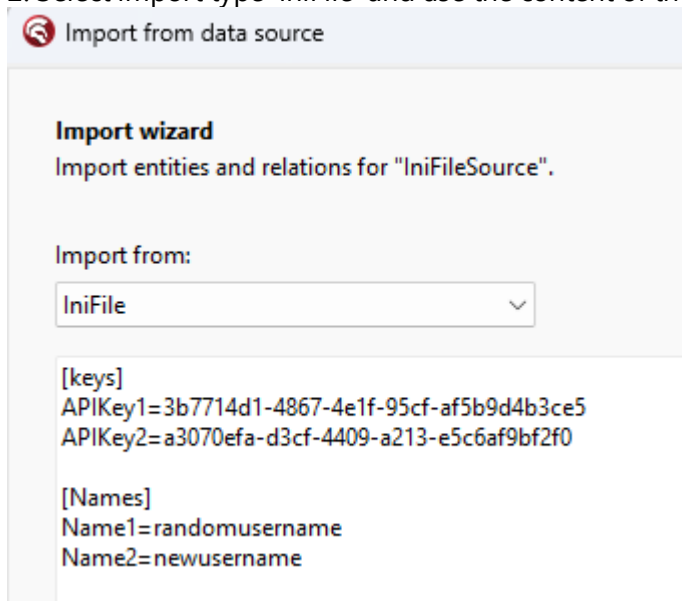
[IniFile Write](#) 131

To create an ini file datasource, follow the next steps

1. Create a new datasource called "IniFileSource" or any other name, select database type memory and set import on true



2. Select import type 'IniFile' and use the content of the ini file as importdata



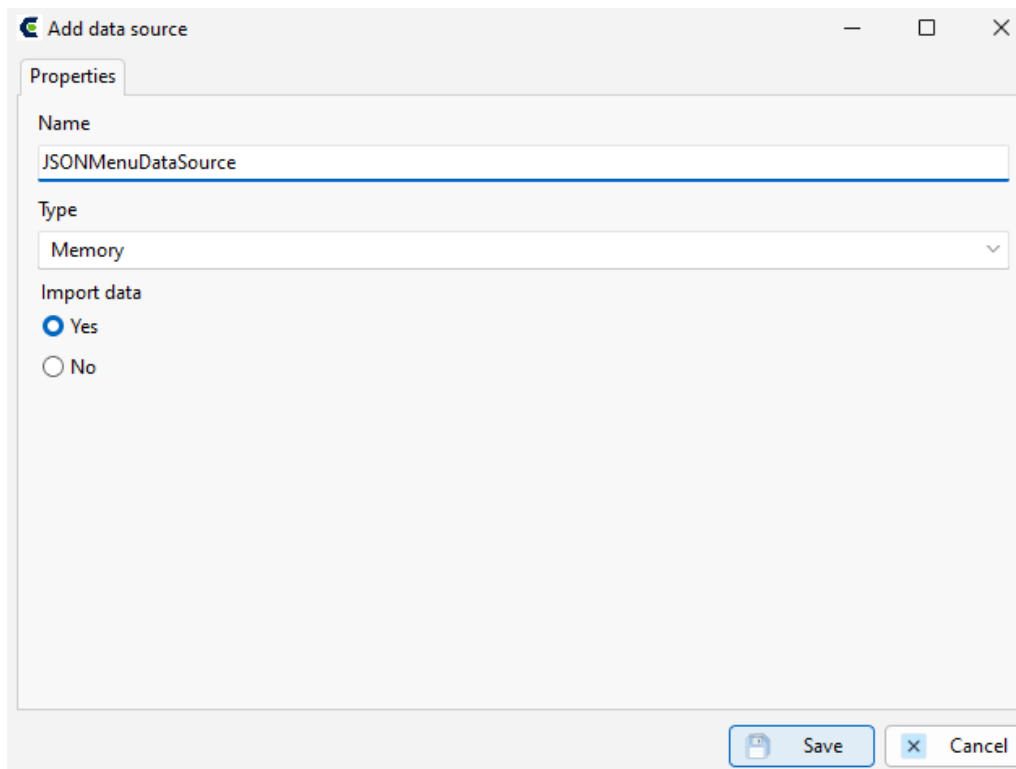
3. Select 'Save'

The ini file datasource is now ready to be used in the ini file activities.


6.8 JSON datasource

A Json datasource is a memory datasource that can be created to handle complex json sturctures.

You can create a new datasource for JSON by adding a memory datasource, and import the structure from JSON.



1. Create new datasource

 Import from data source

Import wizard
Import entities and relations for "JSONMenuDataSource"

Import from:

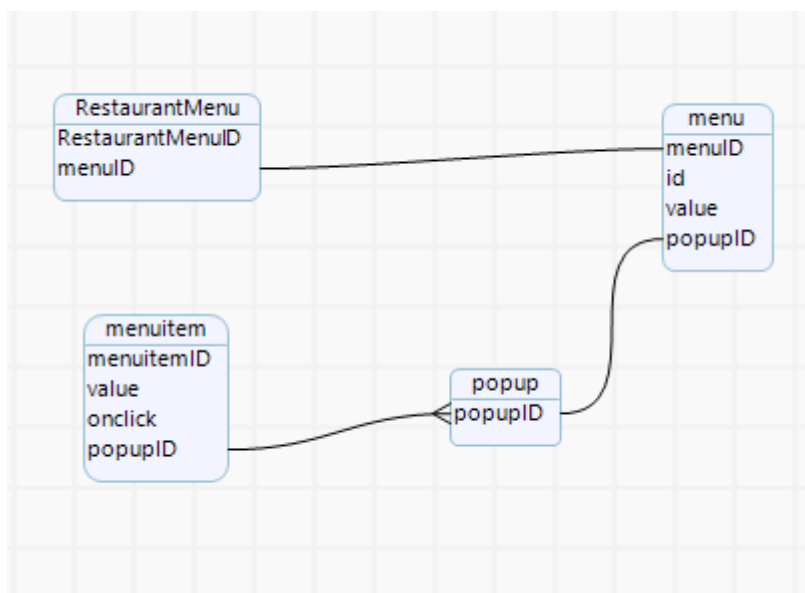
JSON

Entity name:

RestaurantMenu

```
{ "menu": {  
  "id": "file",  
  "value": "File",  
  "popup": {  
    "menuitem": [  
      { "value": "New", "onclick": "CreateNewDoc()" },  
      { "value": "Open", "onclick": "OpenDoc()" },  
      { "value": "Close", "onclick": "CloseDoc()" }  
    ]  
  }  
}  
}
```

2. Import json structure.



3. Resulting data structure

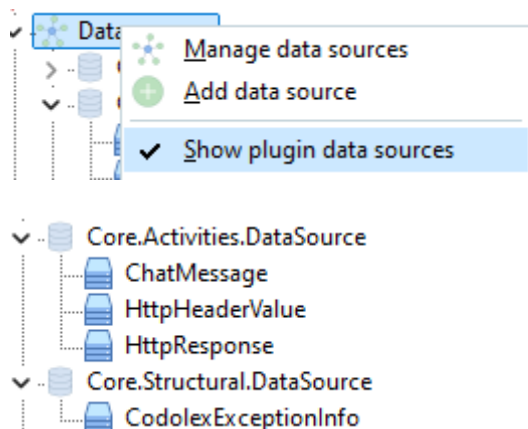
These entities are now Codolex entities available for use in flows.
More information on how to work with JSON:

[Json](#) ¹³²

[Entity conversion](#) ¹⁰²

6.9 Plugin datasources

Plugin datasources are datasources made available in Codolex through activities or third party components.



These datasources contains entities that can be created or returned by activities. These entities can be used like any other entities in Codolex. With exception of saving the entity.

Activities

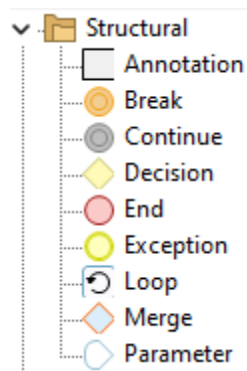
7 Activities

Activities are the building blocks of Codolex. We will cover the most commonly used activities here.

With Codolex it is also possible to create your own activities. Click [here](#)¹⁹⁷ if you want to develop your own activity.

7.1 Structural

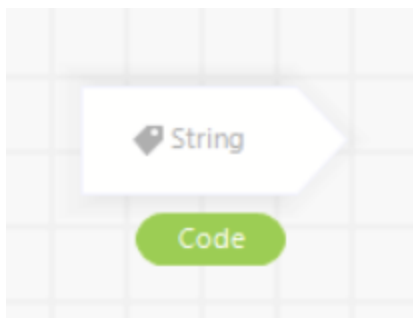
With structural activities within Codolex, you define the 'flow' of the application's logic. Codolex has several ways of modifying the business logic; via decisions, loops, and sequences.



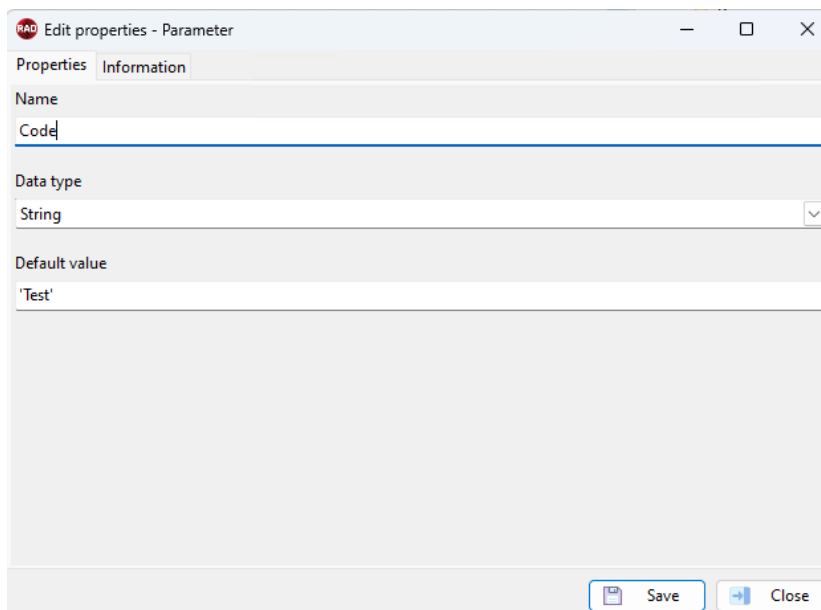
- [Parameters](#)⁵⁷
- [End](#)⁵⁹
- [Start](#)⁶⁰
- [Sequences](#)⁶¹
- [Decision](#)⁶¹
- [Merge](#)⁶²
- [Loop](#)⁶³
- [Exceptions](#)⁶⁵

7.1.1 Parameters

Variables can be passed to flows via parameters. These parameters can be of any type.



Parameter for flow



Edit properties - Parameter

Properties Information

Name

Code

Data type

String

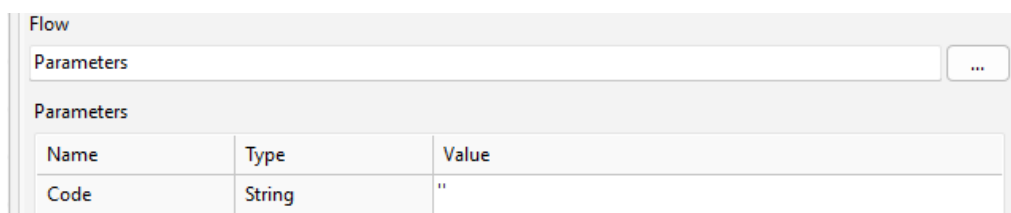
Default value

'Test'

Save Close

```
procedure ParameterFlow(const Code: string = 'Test');
```

When calling a flow from another flow (via the [Flow call](#)⁶⁷ activity) or via Delphi, you need to fill in these parameters.



Flow

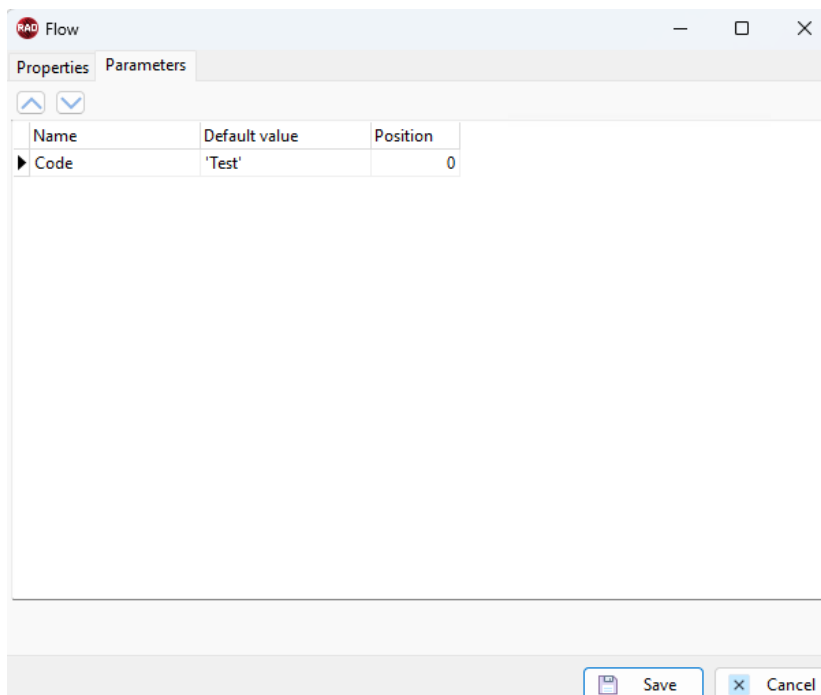
Parameters

Parameters

Name	Type	Value
Code	String	

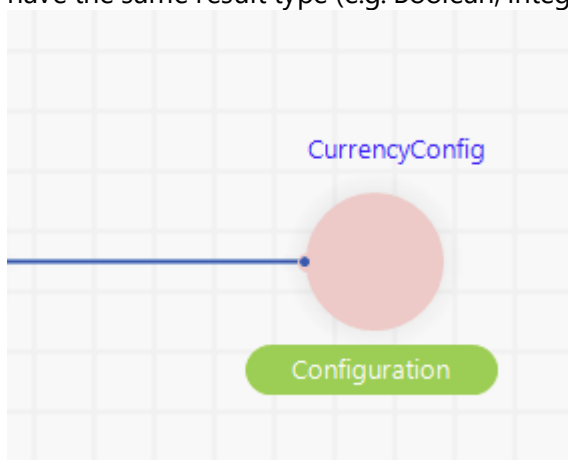
Call flow action

To change the order of the parameters, use the **Parameters** tab on the flow properties screen



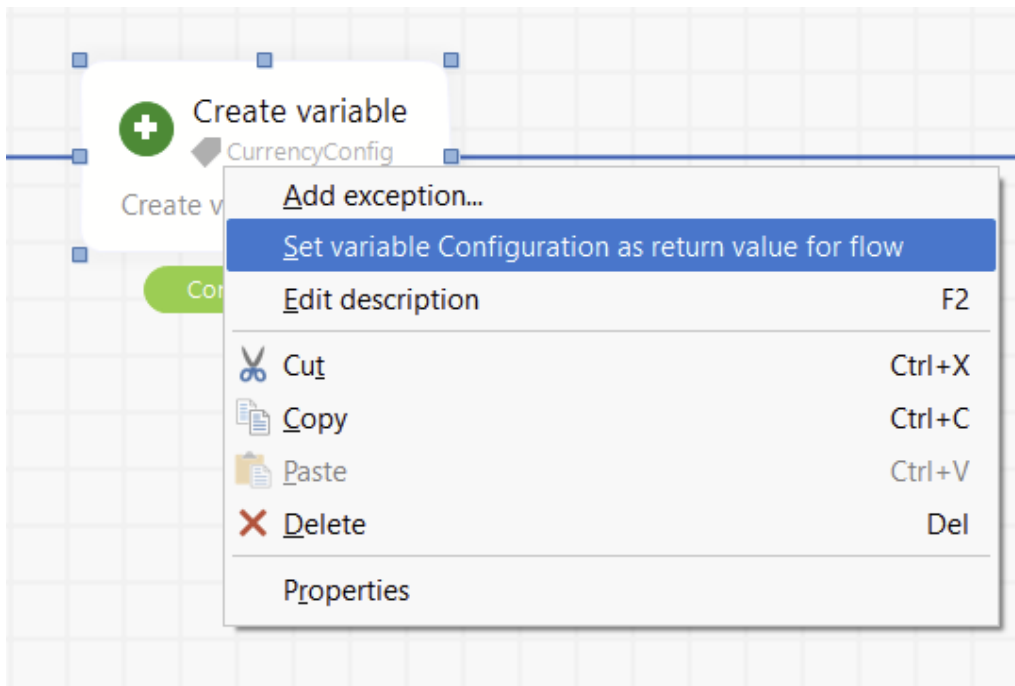
7.1.2 End

A flow can have multiple End activities. The End activity will stop the execution of the flow and can have a result value to set the result of the flow. Every End must have the same result type (e.g. Boolean, Integer, string or custom type).

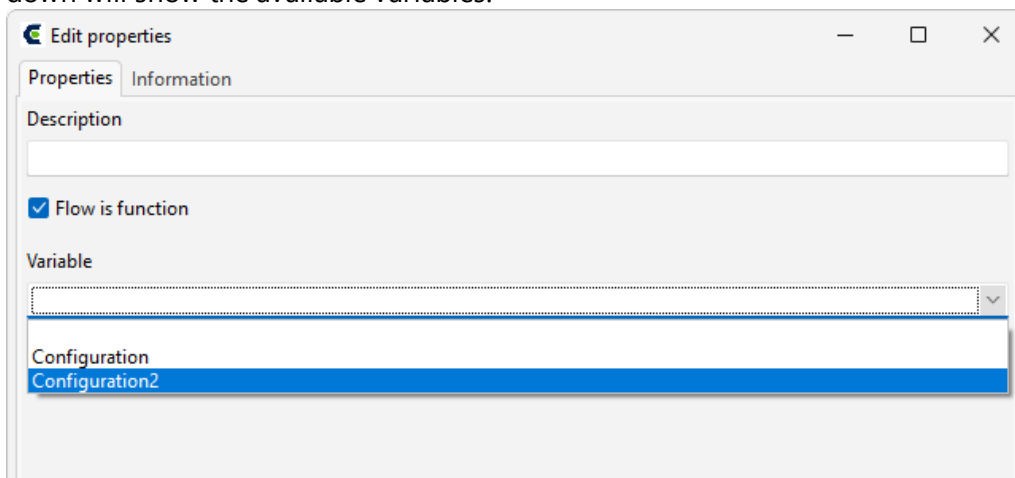


Flow end

To set the flow result, double-click on the End activity, or select "Set variable as return value" on an activity.

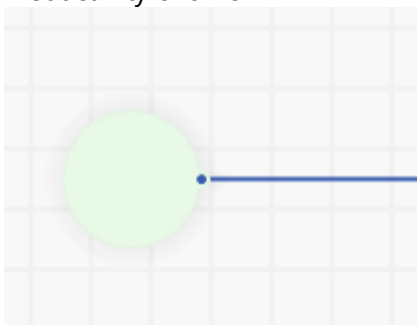


You can also set the result variable in the properties of an end activity. The drop down will show the available variables.



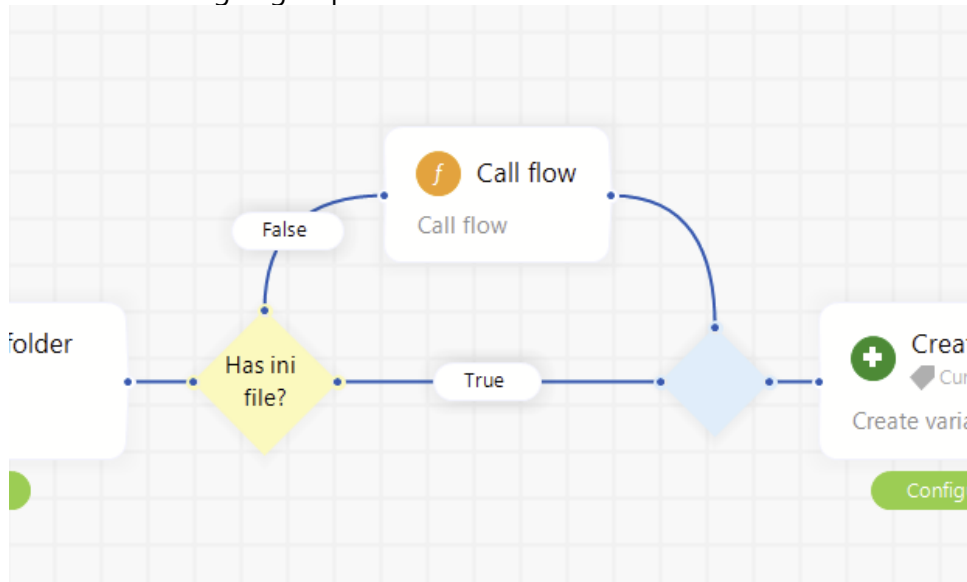
7.1.3 Start

Every flow has exactly one Start activity. This activity is created automatically and can't be deleted from a flow. There is one sequence from the start activity to the first activity of a flow



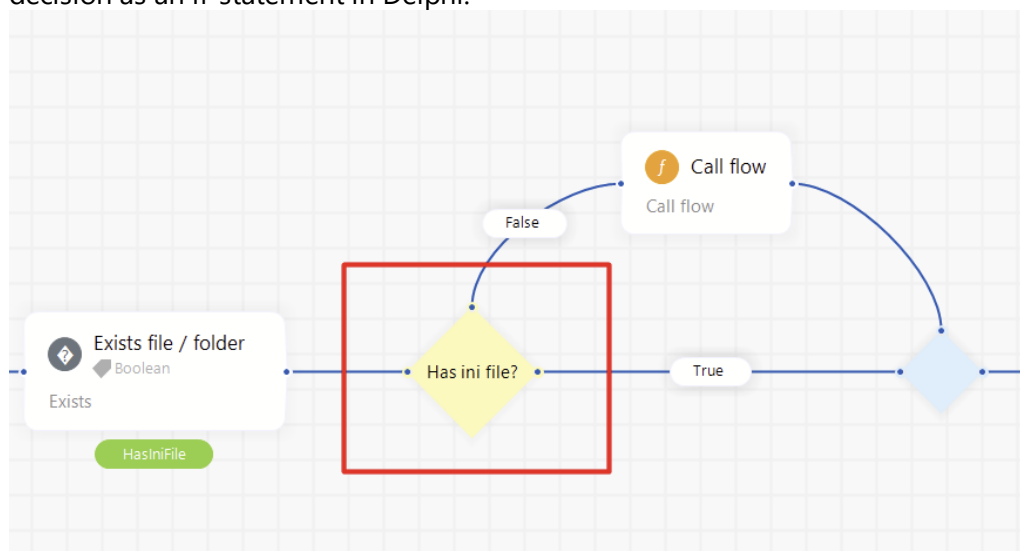
7.1.4 Sequences

A sequence connects individual activities. Sequences allow you to set the order of a flow. An activity always needs one or more incoming sequences and can have one or more outgoing sequences.



7.1.5 Decision

A decision is the structural element to work with the logic of a flow. Think of a decision as an if-statement in Delphi.



Decision in flow

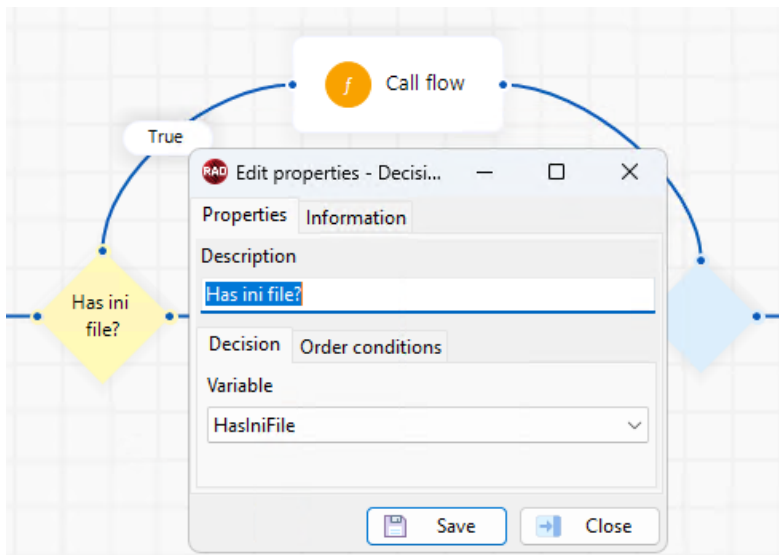
```

begin
  if (not (HasIniFile)) then
  begin
    HelpAndManualScreenshots.Activities.StringUtils.TStringUtils.St
    ringParts();
  end
  else
  
```

```
end; ...
```

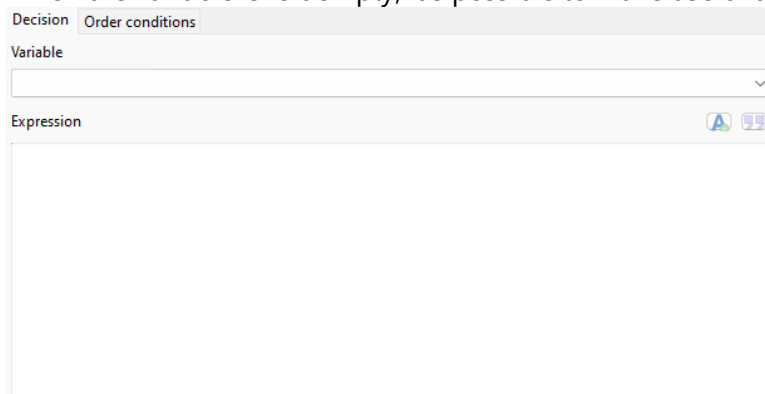
Resulting code

When you double-click a decision, you can set its properties. Then choose the variable on which the decision acts.



Next, set the value of the variable for each outgoing sequence. In this example, the variable `HasIniFile` is a boolean, so there are two possibilities, indicated on the outgoing sequences. Double-click on a sequence to set the sequence value.

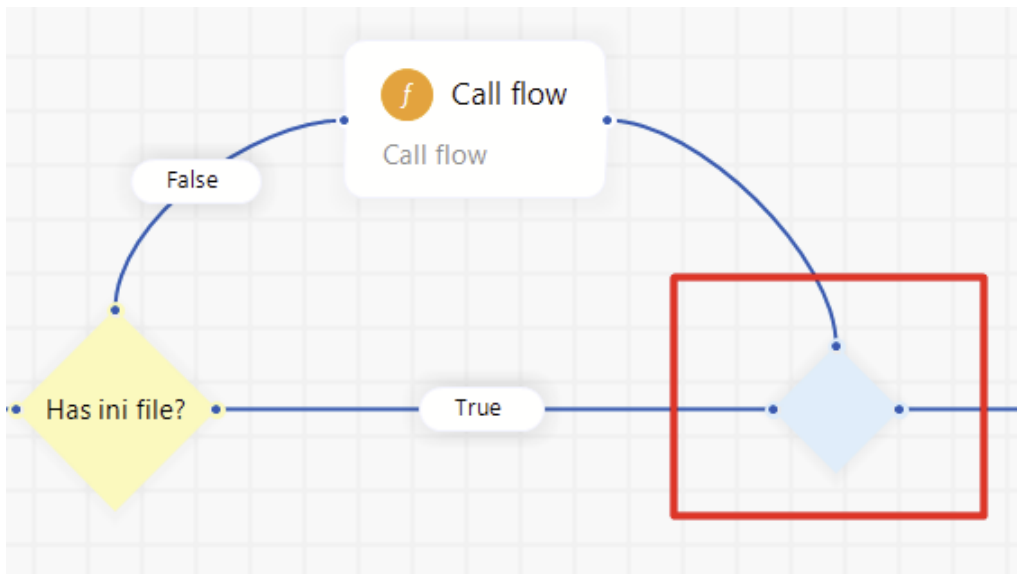
When the variable is left empty, it's possible to make use of an expression.



The result of this expression must always be a boolean.

7.1.6 Merge

The merge activity can be used to bring the different lines within a flow back together. Often, a decision activity is followed up by a merge activity.



7.1.7 Loop

The loop activity offers the ability to iterate over lists. There are three different loop types: While, For, and For..In.

Edit properties

Properties

Description

Loop type

☐ While ☐ For ☒ For..in

Loop variable name

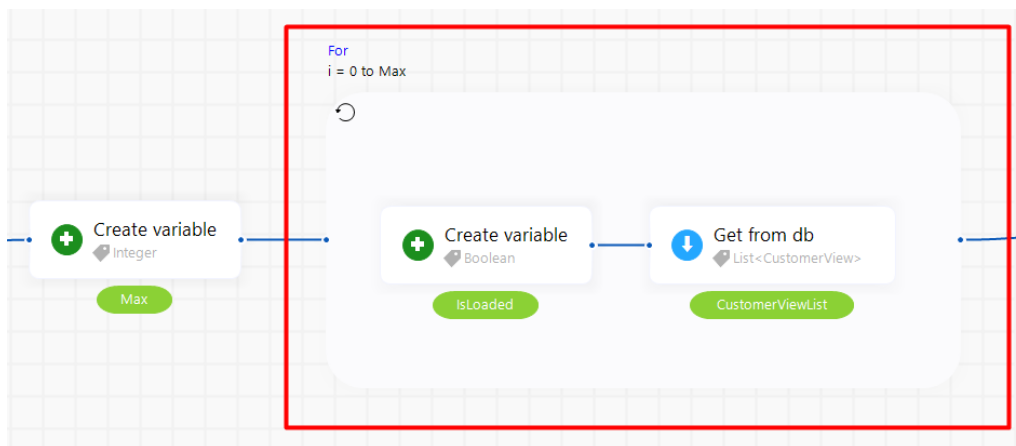
Variable with collection

Save Close

A *Loop variable name* has to be defined for use inside the loop, the item from the list will be parsed in this variable for use.

Variables defined outside the loop are usable inside the loop. However, variables defined inside a loop are **not** usable outside the loop.

Within the loop itself, you have to define the (sub)flows.



Loop in flow

```

begin
  for var i := 0 to Max do
    begin
      var IsLoaded: Boolean;
      IsLoaded := True;
      var CustomerViewList:
      ICodexList<Project.DataSource.Codex.ICustomerView>;
      var SQL :=
        'SELECT CustomerView.* ' + sLineBreak +
        'FROM CustomerView AS CustomerView ';

      var Params: IDatabaseParams := TDatabaseParams.Create;

      CustomerViewList :=
      Project.DataSource.Codex.CustomerViewDataBroker.GetList(SQL,
      Params);
    end;
  end;
end;
  
```

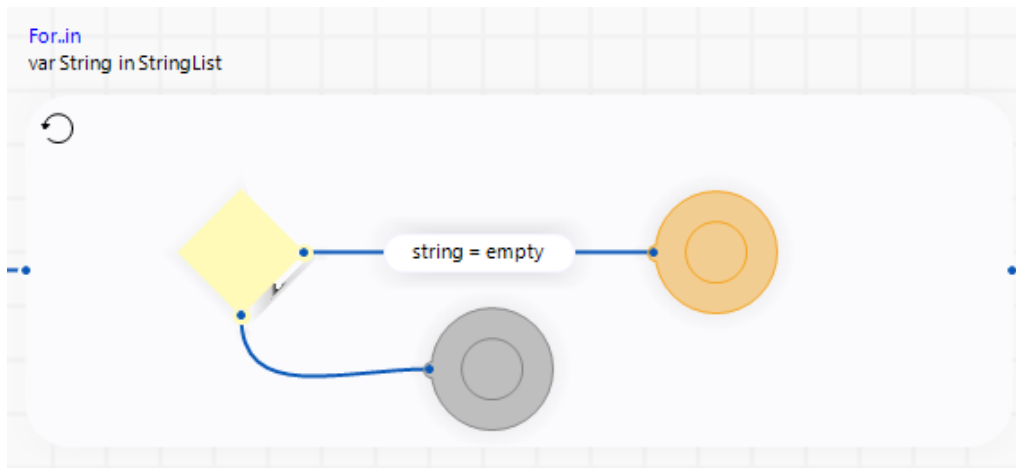
Resulting code

Break and Continue

Inside a loop, the break and continue activities can be used.

A **break** will stop the loop and continue the flow after the loop.

A **continue** will stop this iteration, and go to the next iteration if there is one available



Loop in flow

```

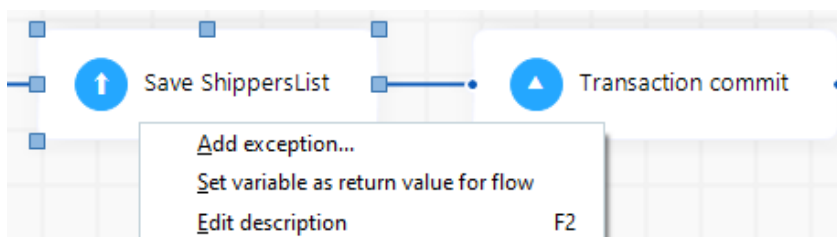
..
for var String in StringList do
begin
  if (String = value = null) then
  begin
    Break;
  end
  else
  begin
    Continue;
  end;
end;
end;
..

```

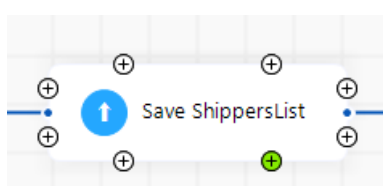
Resulting code

7.1.8 Exceptions

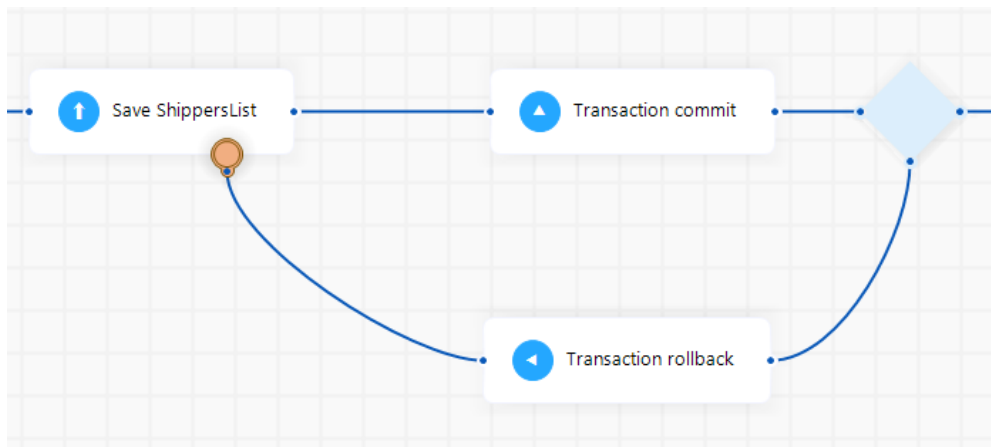
You can use Exceptions to catch possible errors in the application. You can add exception handling to activities in Codalex. To do that, right-click on an activity in a flow, and choose "Add exception".



Then click on one of the + points of the activity to create the exception.



Make sure this point is connected to an alternative flow.



Save with exception handler

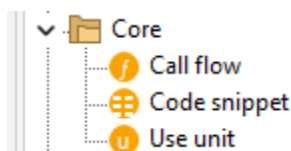
```

begin
  try
    HelpAndManualScreenshots.DataSource.Codolex.ShippersDataBroker.
    Save(ShippersList);
    var Connection :=
    CodolexFramework.DatabaseQueryProvider['Codolex'].Connection;
    Connection.Commit;
  except
    on E: Exception do
      begin
        var Connection1 :=
        CodolexFramework.DatabaseQueryProvider['Codolex'].Connection;
        Connection1.Rollback;
      end;
    end;
  end;
end;
  
```

Resulting code

7.2 Core

The core activities provide the basis for handling flows, and supply them with units of extra code



- [Flow call](#) ⁶⁷
- [Code snippet](#) ⁶⁸
- [Use unit](#) ⁶⁸

7.2.1 Flow call

A flow call can be used to call another flow inside a flow.

To use the call flow activity, the activity can be dragged into a flow.

Another option is to drag a flow from the Project Explorer, this will create a call flow activity with the dragged flow selected.

Name	Type	Value
Code	String	"
Dates	List<DateTime>	DatesList
Shippers	Shippers	Shippers

Call flow

Begin

```
CoreActivities.Parameters(' ', DatesList, Shippers);  
end;
```

The parameters defined in the flow that is being called are listed in the call flow properties.

The value of the parameters need to be defined to prevent errors. The value can be static values or variables.

The class instance to use is optional for providing a class instance, this is mandatory if the flow is from another flowclass

If the flow has a return variable, a return value field appears, where the name of the result variable can be defined.

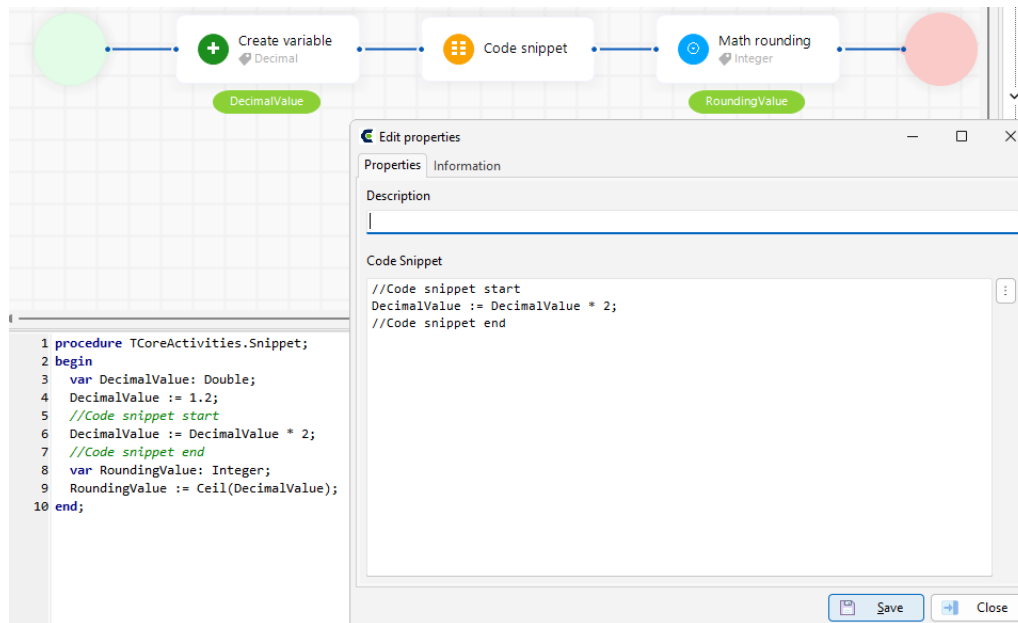
Return value
ParametersResultList

Tip, always keep an eye on which flow calls which flow. It is possible to create a loop, that will result in a stack overflow...

7.2.2 Code snippet

Should a developer encounter the situation that Codolox lacks functionality which is available in Delphi, the code snippet can be used.

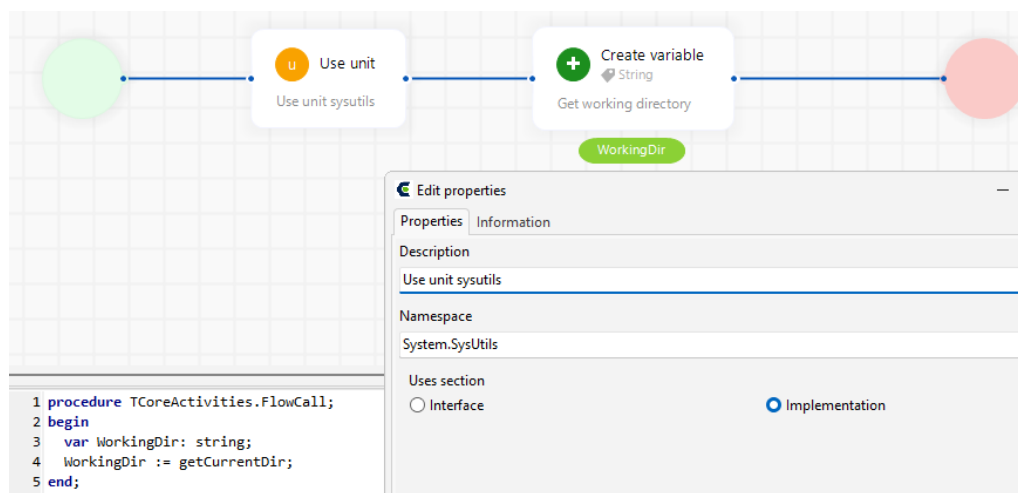
The code snippet parses code directly into the generated flow.



7.2.3 Use unit

Important sections of the Delphi class are the uses sections (interface and implementation).

In the case that you need code from another unit available in Delphi, the use unit provides the possibility to add a unit to the uses section.

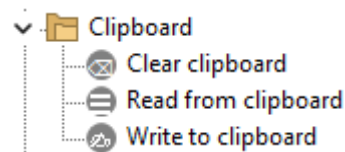


Example: Get the working directory with the function `getCurrentDir` from `System.Sysutils`

Depending on what you need, you can select if the use must be added to interface or implementation section. By default implementation is selected

7.3 Clipboard

A few activities that helps in getting value from or to the clipboard

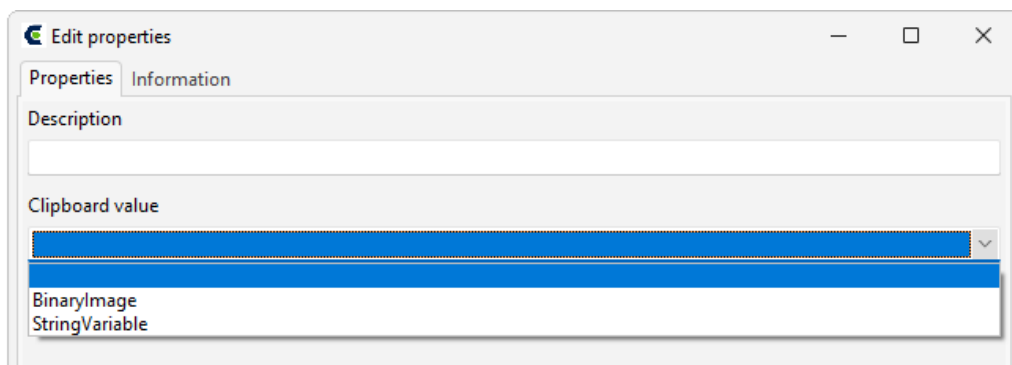


- [Clear clipboard](#) ⁷¹
- [Read from clipboard](#) ⁷⁰
- [Write to clipboard](#) ⁶⁹

7.3.1 Write to clipboard

Write a value to the users clipboard with this activity

Uses [Vcl.Clipbrd.TClipboard](#)



Both a string and image can be selected

```
begin
  var ClipBoard := TClipboard.Create;
  try
    ClipBoard.Open;
    ClipBoard.AsText := StringVariable;
  finally
    ClipBoard.Close;
    ClipBoard.Free;
  end;
end;
```

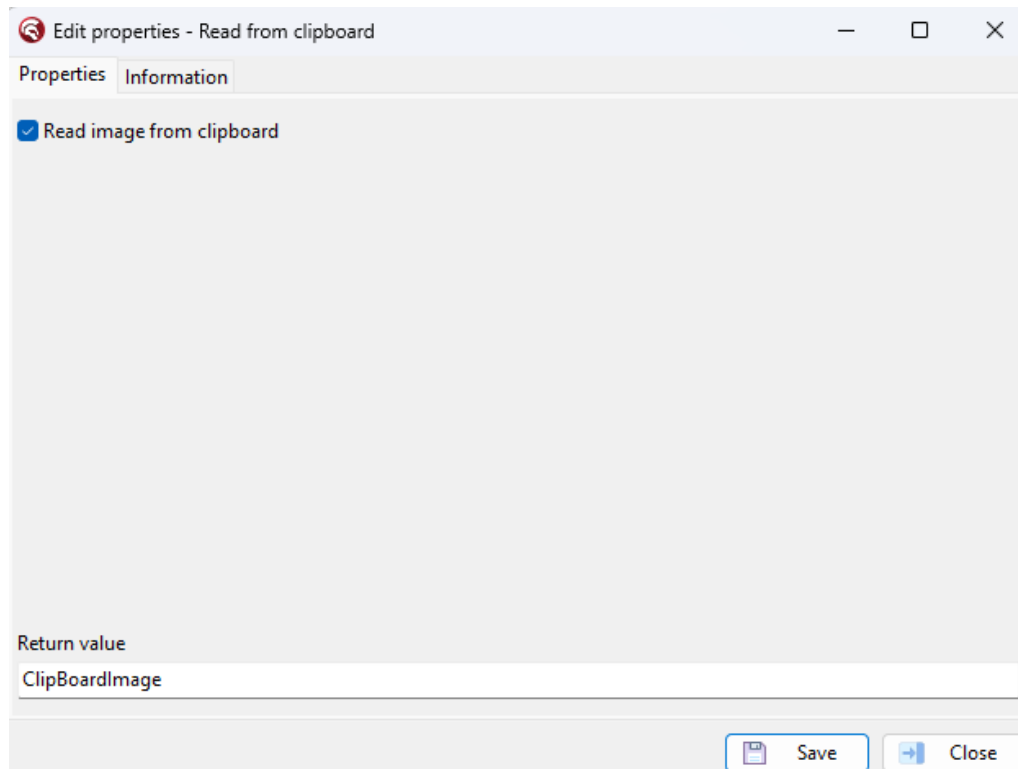
Resulting code when string is selected

7.3.2 Read from clipboard

The read from clipboard value can be used to get a string or an image from the clipboard

Uses [Vcl.Clipbrd.TClipboard](#)

By default a string is expected. To get an image, select the *read image from clipboard* option. This will result in a binary variable.



Activity properties

```
begin
  var ClipboardImage: ICodolexBinary;
  var Clipboard := TClipboard.Create;
  try
    Clipboard.Open;

    var Image := TImage.Create(nil);
    var MemoryStream := TMemoryStream.Create;
    try
      ClipboardImage := TCodolexBinary.Create;
      var Picture := Image.Picture;
      if Clipboard.HasFormat(CF_PICTURE) then
        begin
          Picture.Assign(Clipboard);
          Picture.SaveToStream(MemoryStream);
          ClipboardImage.Stream.LoadFromStream(MemoryStream);
        end
      else if Clipboard.HasFormat(CF_BITMAP) then
        begin
```

```

    var BitMap := Picture.BitMap;
    BitMap.Assign(ClipBoard);
    BitMap.SaveToStream(MemoryStream);
    ClipBoardImage.Stream.LoadFromStream(MemoryStream);
  end;
finally
  Image.Free;
  MemoryStream.Free;
end;
finally
  ClipBoard.Close;
  ClipBoard.Free;
end;
end;

```

7.3.3 Clear clipboard

Use the activity to clear the clipboard. this can be helpful if multiple values are being copied but you want to avoid flooding the clipboard of the user

Uses [Vcl.Clipbrd.TClipboard](#)

The activity has no properties to fill.



Activity in flow

```

..
var ClipBoard := TClipboard.Create;
try
  ClipBoard.Open;
  ClipBoard.Clear;
finally
  ClipBoard.Close;
  ClipBoard.Free;
end;
..

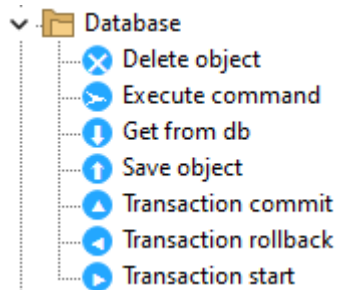
```

Resulting code

7.4 Database

Codolex has an extensive entity framework. Although it is not mandatory to use this within Codolex, this entity framework makes working with Codolex much

faster. In addition, it is possible to use this framework in conjunction with a legacy code structure. The database activities need a datasource with a database connection to perform actions on a database.



- [Get from DB](#) 72
- [Save object](#) 74
- [Delete object](#) 74
- [Transactions](#) 75
- [Execute command](#) 78

7.4.1 Get from DB

The "Get from DB" activity is a convenient way to quickly retrieve data from a database, independent of the type of database. The result of this activity is a list of entities, which can be used in the flow. For example, this can be used in the loop activity or the List Operation activities.

Activity properties

```

begin
  var CustomersList:
  ICodoloxList<HelpAndManualScreenshots.DataSource.Codolox.ICustomers
  >;
  var SQL :=
    'SELECT Customers.* ' + sLineBreak +
    'FROM Customers AS Customers ';

  var Params: IDatabaseParams := TDatabaseParams.Create;

  CustomersList :=
  HelpAndManualScreenshots.DataSource.Codolox.CustomersDataBroker.Get
  List(SQL, Params);
end;

```

Resulting code

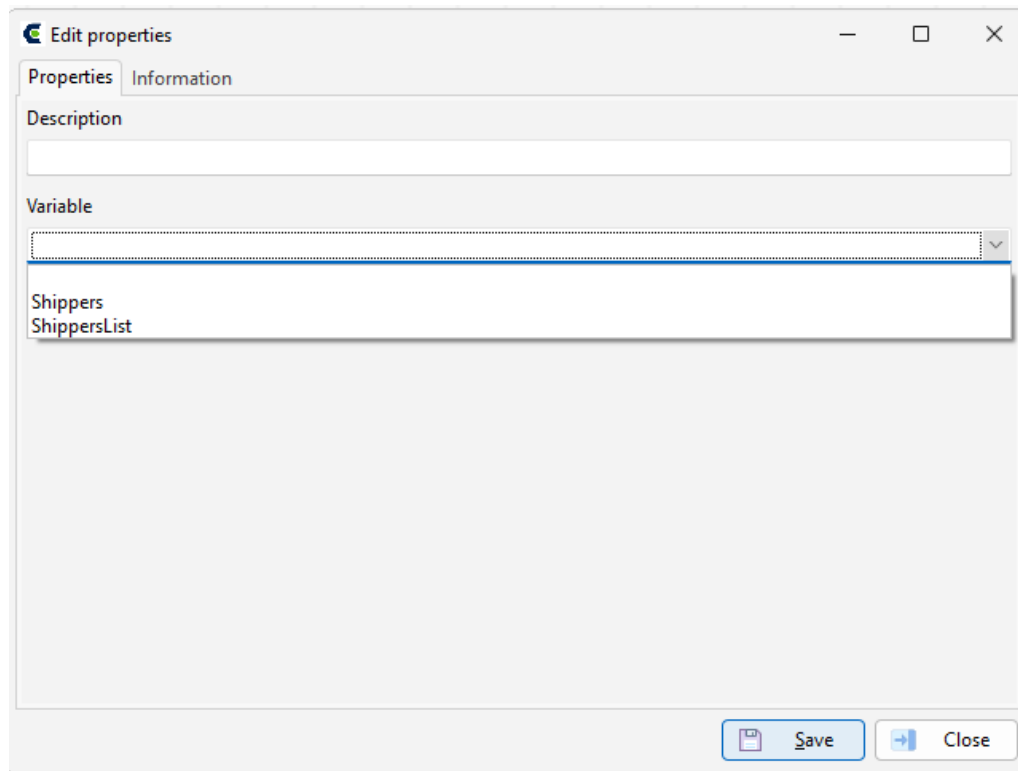
To get started, select an entity from the search list. Codolox will fill in the basic select statement. Using the "where, order by and limit results tabs" you can specify the exact statement. The return value will be a list, or a single entity if you select "first" at the Limit results tab.

7.4.2 Save object

Save object to the database with the save object activity.

A variable entity can be selected to save.

Without transactions, the saved object is directly persistent in the database.



Activity properties

```
begin
    HelpAndManualScreenshots.DataSource.Codollex.ShippersDataBroker.Save(Shippers);
end;
```

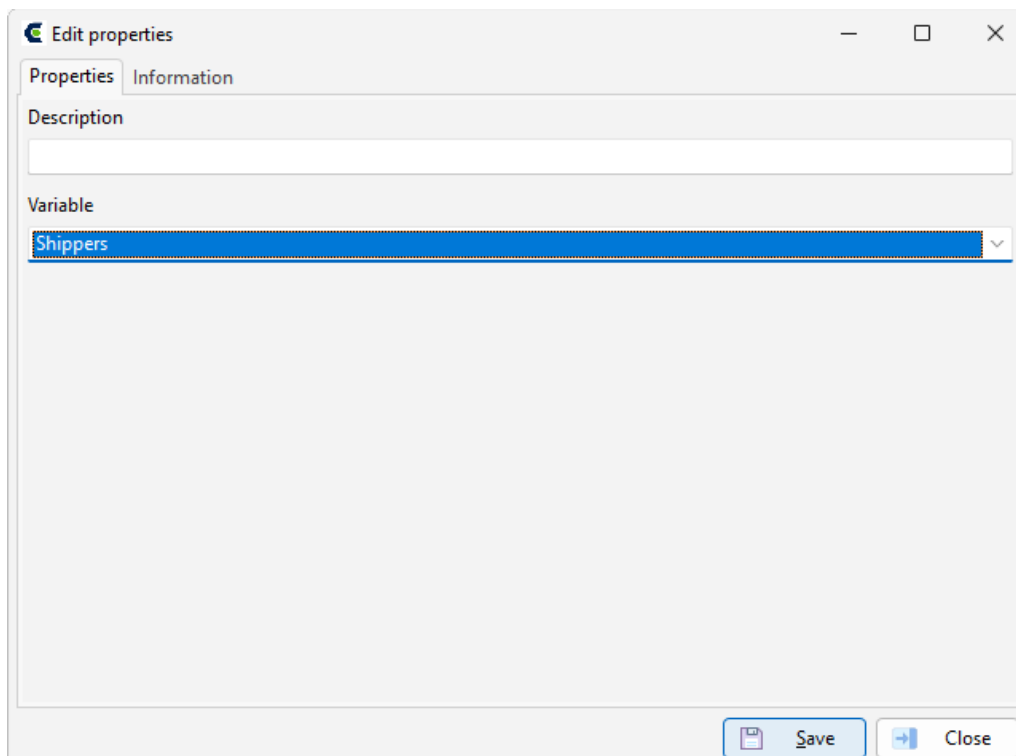
Resulting code when shippers selected

When saving a new object (e.g. through create variable), the entity is translated to a new record.

It's also possible to provide a list of entities instead of a single entity. All entities in the list will be saved.

7.4.3 Delete object

Delete object that is present in a database.



Activity properties

```
begin
    HelpAndManualScreenshots.DataSource.Codolex.ShippersDataBroker.Delete(Shippers);
end;
```

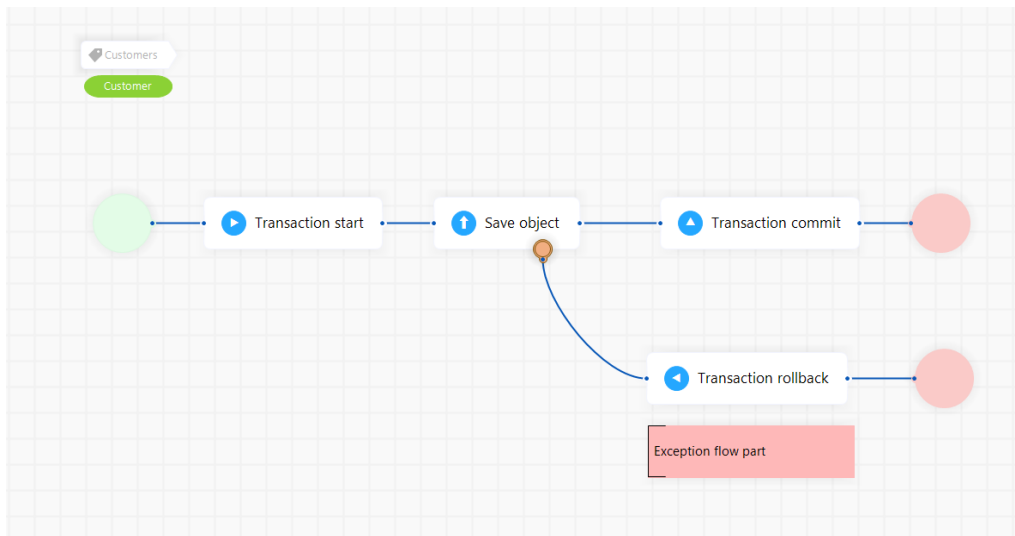
Resulting code

If an object is not present in a database, when parsing or creating entities for example, this activity will not result in a change.

This activity will not free an object. To free an object from memory, see [\[reference\]](#) for more information.

7.4.4 Transactions

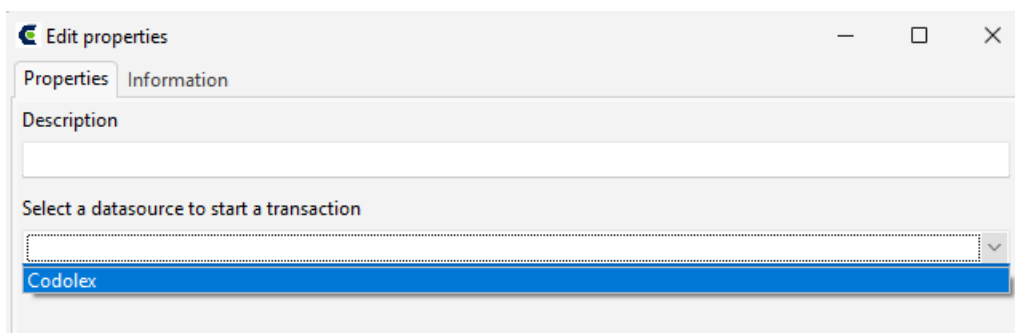
Transactions provide a secure way of working with databases. There are three possible activities: start, commit and rollback. Use these together with [Exceptions](#)⁶⁵ to ensure data is saved correctly.



- [Start transaction](#) ⁷⁶
- [Commit transaction](#) ⁷⁶
- [End transaction](#) ⁷⁷

7.4.4.1 Start transaction

A transaction on a database can be started by the 'start transaction' activity.



Activity properties

```

begin
  var Connection :=
    CodolexFramework.DatabaseQueryProvider['Codolex'].Connection;
  Connection.StartTransaction;
end;
  
```

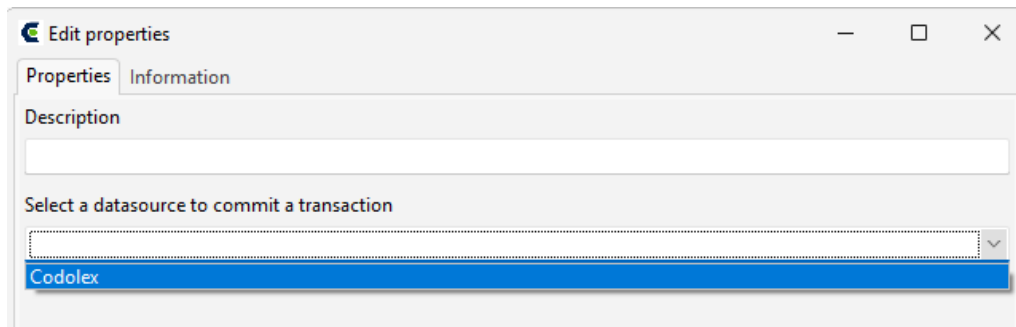
Resulting code

Select one of the available databases that can work with transactions in the properties.

It is possible to start multiple transactions on one database.

7.4.4.2 Commit transaction

The 'Transaction commit' activity commits the database changes made in between the start of a transaction and this activity.



Activity properties

```
begin
  var Connection :=
    CodolexFramework.DatabaseQueryProvider['Codolex'].Connection;
  Connection.Commit;
end;
```

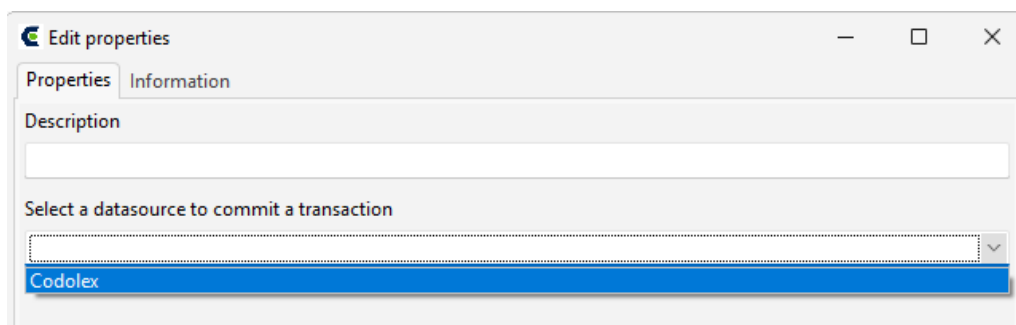
Resulting code

Select one of the available databases that can work with transactions in the properties.

When multiple transactions are started on a database, the last transaction will be committed by this activity.

7.4.4.3 Rollback transaction

The 'Transaction rollback' activity rolls back the database changes made in between the start of a transaction and this activity.



Activity properties

```
begin
  var Connection :=
    CodolexFramework.DatabaseQueryProvider['Codolex'].Connection;
  Connection.Rollback;
end;
```

Resulting code

Select one of the available databases that can work with transactions in the properties.

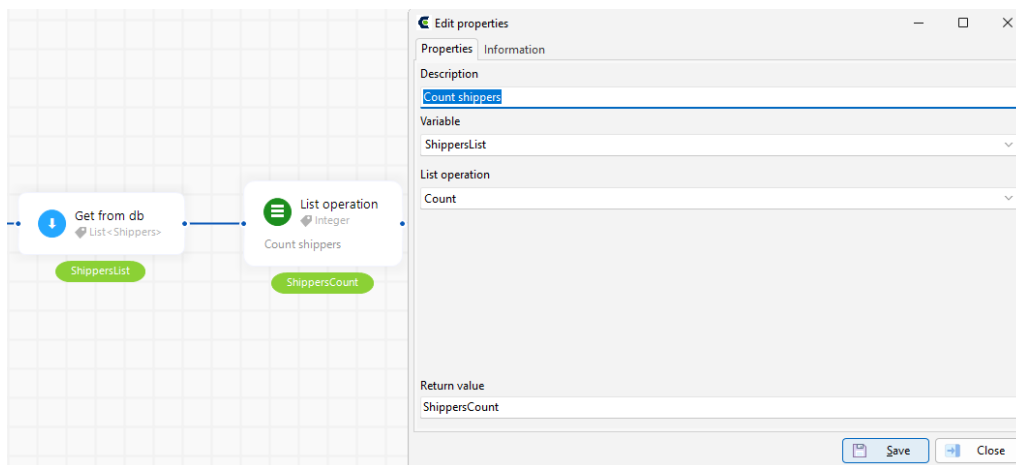
When multiple transactions are started on a database, the last transaction will be rolled backed by this activity.

7.4.5 Execute command

The execute command helps with retrieving data from the database faster. The activity makes it possible to run a command on the database directly without retrieving data.

This can be used to retrieve single fields of primitive data types.

Let's compare the method for retrieving a record count for example. You could get the amount of records in a table by retrieving all records, and using the list operation activity.



In the case you only want to know the count, and not perform any other action with the records, this solution would be performance heavy because it retrieves all records.

The other option is to execute a count action on the database with the Execute command activity.

Edit properties - Execute command

Properties Information

Database command to execute

'SELECT COUNT(*) as shipperscount FROM shippers'

Datasource to run database command

Codolex

Result field data type

Integer

Result field name

shipperscount

Return value

DatabaseCommandResult

Save Close

Activity properties

```
begin
  var shipperscount: Integer;
  var DatabaseQuery :=
    CodolexFramework.DatabaseQueryProvider['Codolex']();
  DatabaseQuery.SQL.Text := 'SELECT COUNT(*) as shipperscount FROM
shippers';
  var Dataset := DatabaseQuery.Open;
  Dataset.First;
  var Field := Dataset.FindField('shipperscount');
  if Assigned(Field) then
    shipperscount := Field.AsInteger;
end;
```

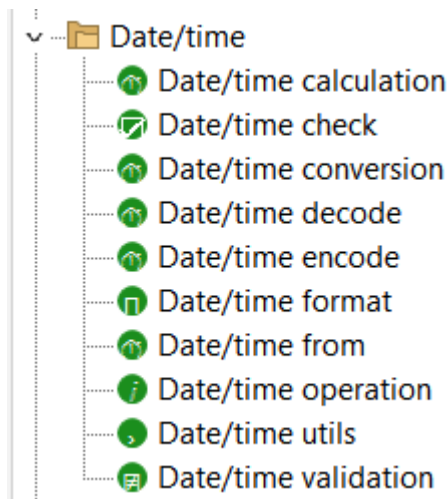
Select a database, provide the command to execute, and define what the result field name and type will be.

The activity will look for this result field and put the value in a result variable.

If multiple records are returned from the command, the first record will be taken for the result.

7.5 Date/Time

The various activities in the Date/Time category can help you work with date and time calculations, comparisons and conversions.



[Calculations](#)  80

[Check](#)  82


[Conversion](#)  83

[Decode](#)  85

[Encode](#)  86

[Format](#)  87

[From](#)  88

[Operation](#)  89

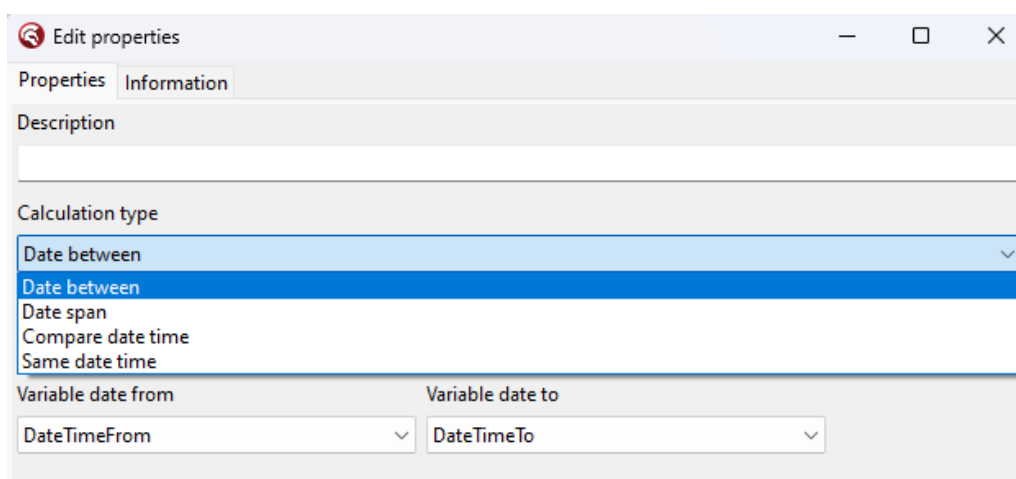
[Utils](#)  91

[Validation](#)  92

7.5.1 Calculations

Date time calculations can be used to compare two dates, this can be done in 4 ways.

Uses [System.DateUtils](#)



[Date Between](#) +

Returns the amount of [timeframe] as integer between the 2 dates. The number is always rounded down.

Available time frames are **year, month, week, day, hours, minute, second, millisecond**.

Example:

Time frame: **week**, Date from: **2024-01-01**, Date to: **2024-02-23**

```
begin
  CompareTime := WeeksBetween(DateTimeFrom, DateTimeFrom);
  var DialogResult: Integer;
end;
```

Result: 7

[Date Span](#) +

Date span works the same as date between, only return a double rounded down to 2 decimals.

Example:

Time frame: **week** - Date from: **2024-01-01** - Date to: **2024-02-23**

```
begin
  var CompareTime: Double;
  CompareTime := DaySpan(DateTimeFrom, DateTimeFrom);
end;
```

Result: 7.64

[Compare Date Time](#)

The compare calculation does not return an amount of something in between, but only a positive or negative integer based on which date is older.

If the date from is later than the date to, the result is positive.

Example:

Date from: **2024-01-01** - Date to: **2024-02-23**

```
begin
  var CompareTime: Integer;
  CompareTime := CompareDateTime(DateTimeFrom, DateTimeFrom);
end;
```

Result: -1

Date from: **2024-02-23** - Date to: **2024-01-01**

```
begin
  var CompareTime: Integer;
  CompareTime := CompareDateTime(DateTimeFrom, DateTimeFrom);
end;
```

Result: 1

If the results are the same, the result will be 0

[Same Date Time](#)

If you want to know if a date time is the same, and have the result in a directly boolean, use the **Same date time** calculation.

Example:

Date from: **2024-01-01** - Date to: **2024-01-01**

```
begin
  var CompareTime: Boolean;
  CompareTime := SameDateTime(DateTimeFrom, DateTimeFrom);
end;
```

Result: true

7.5.2 Check

The Check activity can be used to validate a few things about a date time value. The return value will always be a boolean.

Uses [System.DateUtils](#)

Datetime value
DateToCheck
Check to preform
AM
PM
InLeapYear
SameDay
Today

[IsAM](#)

This check will validate if the given date time is in AM time range. 00:00 - 11:59 in 24hour notation.

Example:

Date to check: **2024-01-01**

```
begin
  var Variable: Boolean;
  Variable := IsAM(DateToCheck);
end;
```

Result: True

[IsPM](#)

This check will validate if the given date time is in PM time range. 12:00 - 23:59 in 24hour notation.

Example:

Date to check: **2024-01-01**

```
begin
  var Variable: Boolean;
  Variable := IsPM(DateToCheck);
end;
```

Result: False

[InLeapYear](#)

This check only looks at the year in a given date time, and will tell you if it's in a leap year or not.

Example:

Date to check: **2024-01-01**

```
begin
  var Variable: Boolean;
  Variable := IsInLeapYear(DateToCheck);
end;
```

Result: True

[SameDay](#)

This check will validate if it is the same day as another date. It has to be the exact day (year, month, day). Time on the day does not matter.

Example:

Date to check: **2024-01-01** - Check date value: **2024-01-08**

```
begin
  var Variable: Boolean;
  Variable := IsSameDay(DateToCheck, SameDateToCheck);
end;
```

Result: False

s

[Today](#)

This check will validate if it is the same day as today, so the outcome will be dependent on the day the program runs.

Example:

Date to check: **2024-01-01**

```
begin
  var Variable: Boolean;
  Variable := IsToday(DateToCheck);
end;
```

Result: False

7.5.3 Conversion

This activity converts a datetime variable into a string or integer variable, depending on the method.

Uses [System.DateUtils](#)

Description

Datetime value

Date

Convert to

Unix

☐ Use UTC time

Return value

ConvertedTime

Save Close

There are 5 options to convert to: **Unix, ISO8601, JulianDate, ModifiedJulianDate, Milliseconds.**

The 'Use UTC time' option is available on some conversions to indicate to the activity if the date provided is UTC, or the local time zone of the machine.

Unix

Returns an integer value of milliseconds between the given date and 1970-01-01 00:00:00 UTC.

Example:

Datetime value: **2024-01-01 00:00:00** - Use UTC time: **true**

```
begin
  var ConvertedTime: Int64;
  ConvertedTime := DateTimeToUnix(Date, True);
end;
```

Result: 1704067200

ISO8601

Returns a string of the date formatted following ISO8601 standards.

Example

Datetime value: **2024-01-01 00:00:00** - Use UTC time: **false** - Time zone: **UTC+1**

```
begin
  var ConvertedTime: string;
  ConvertedTime := DateToISO8601(Date, False);
end;
```

Result: 2025-01-01T00:00:00.000+01:00

JulianDate

The Julian date is the number of days, including fractional days, since 4713 BC January 1, Greenwich noon.

Example:

Datetime value: **2024-01-01 00:12:00**

```
begin
  var ConvertedTime: Double;
  ConvertedTime := DateTimeToJulianDate(Date);
end;
```

Result: 2.460.311,00

ModifiedJulianDate

The modified Julian date is the number of days, including fractional days, since Greenwich midnight on November 17, 1858. Modified Julian dates are based on Julian dates, but adjusted to use midnight rather than noon as a starting point. They use a more recent date as a starting point.

Example

Datetime value: **2024-01-01 00:12:00**

```
begin
  var ConvertedTime: Double;
  ConvertedTime := DateTimeToModifiedJulianDate(Date);
end;
```

Result: 60.310,50

Milliseconds

The amount of milliseconds between the given date and 00-00-00 00:00:00 UTC (0)

Example

Datetime value: **2024-01-01 00:00:00**

```
begin
  var ConvertedTime: Int64;
  ConvertedTime := DateTimeToMilliseconds(Date);
end;
```

Result: 63839750400000

7.5.4 Decode

The decode datetime activity provides the possibility to get specific part(s) of a date into integer(s).

Uses [System.DateUtils.DecodeDateTime](#)

Edit properties - Date/time decode

Properties Information

Input variable date time

Date

Output variable year Output variable month Output variable day

YearVar

Output variable hour Output variable minute Output variable second

Save Close

Activity properties

```
begin
  DecodeDateTime(Date, YearVar1, Month1, DayVar1, Hour1, Minute1,
    Second1, MilliSecond1);
  YearVar := YearVar1;
  DayVar := DayVar1;
end;
```

Resulting code

The output variables are optional integer variables. After the activity the variable will be filled with the corresponding value

7.5.5 Encode

The encode activity provides the option to create a date time variable and set the values with integers.

Uses [System.DateUtils.EncodeDateTime](#)

Edit properties - Date/time encode

Properties Information

Year 2024 Month 01 Day 01

Hour 00 Minute 00 Second 00

Return value

Date

Save Close

Activity properties

```
begin
  var Date: TDateTime;
  var Year := 2024;
  var Month := 01;
  var Day := 01;
  var Hour := 00;
  var Minute := 00;
  var Second := 00;
  var MilliSecond := 0;
  Date := EncodeDateTime(Year, Month, Day, Hour, Minute, Second,
    MilliSecond);
end;
```

Resulting code

The input fields are optional with the limit that one must be filled.
If no input is given, 0 is the default value.

7.5.6 Format

Format the date time into a string. The format can be customized with a string and settings.

Uses [System.SysUtils.FormatDateTime](#)

Activity properties

```
begin
  var Variable: string;
  Variable := FormatDateTime('yyyy-MM-dd hh:mm:ss', DateValue);
end;
```

Resulting code

Format

A value that represents the string output. It can be designed to your preferences with the following options:

[Embarcadero wiki FormatDateTime](#)

FormatSettings value

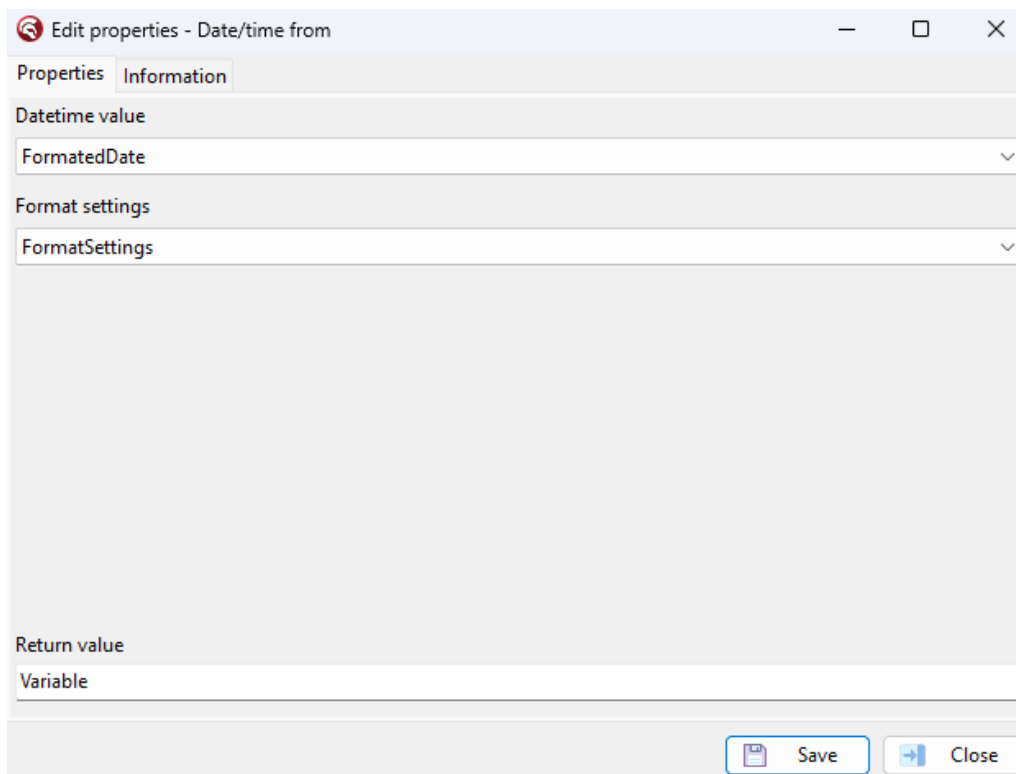
An optional input to provide datetime format settings for this format.

Variable type: [TFormatSettings](#)

7.5.7 From

The date time from activity converts a string to a date time with the current local date/time format.

Uses [System.SysUtils.StrToDateTime](#)



Activity properties

```
begin
  var Variable: TDateTime;
  Variable := StrToDateTime(FormatedDate, FormatSettings);
end;
```

Resulting code

Format settings can be used to deviate from the current local date/time format.
Variable type: [TFormatSettings](#)

7.5.8 Operation

Operations can be used to alter a date time and return a value or a new variable.

Uses [System.DateUtils](#)

Uses [System.SysUtils](#)

Increase +

Increases a date with a given value for a given time frame.

Available time frames: **year, month, week, day, hour, minute, second**

The amount that needs to be changed must be an integer value

Example: Date value: **2023-11-01 00:00:00** - Time frame: **month** - Increase number: **2**

```
begin
  var IncreasedDate: TDateTime;
  IncreasedDate := IncMonth(Date, 2);
end;
```

Result: 2024-01-01 00:00:00

To decrease a date value, use the increase operation with a negative number.

Start Of +

Get the start of the given time frame of the given date time.

Available time frames: **year, month, week, day, hour, minute, second**

Example:

Date value: **2024-12-31 02:04:10** - Time frame: **week**

```
begin
  var StartOfDate: TDateTime;
  StartOfDate := StartOfTheWeek(Date);
end;
```

Result: 2024-12-30 00:00:00

EndOf +

Get the end of the given time frame of the given date time.

Available time frames: **year, month, week, day, hour, minute, second**

Example:

Date value: **2023-07-23 02:04:10** - Time frame: **year**

```
begin
  var EndOfDate: TDateTime;
  EndOfDate := EndOfTheYear(Date);
end;
```

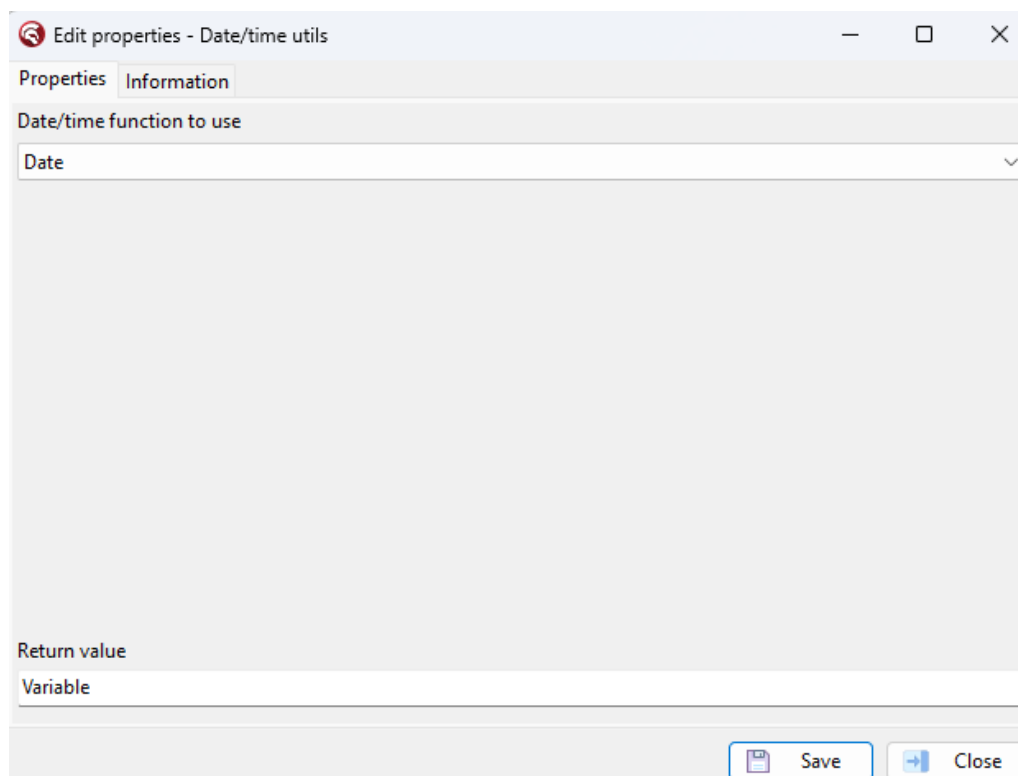
Result: 2023-12-31 23:59:59

7.5.9 Utils

Date time utils are onetime functions that helps with getting datetime standards like now or tomorrow.

Uses [System.DateUtils](#)

Uses [System.SysUtils](#)



Activity properties

```
begin
  var Variable: TDateTime;
  Variable := Date;
end;
```

Resulting code

Date

Gets the current date without the time of the day.

Now

Gets the current date time to the millisecond.

Time

Gets the current time stamp without the date part.

Tomorrow

Gets the start of tomorrow as date time.

Yesterday

Gets the start of the day from yesterday as date time.

7.5.10 Validation

This activity can be used to validate if a date is valid when given integer values for its parts.

Uses [System.DateUtils](#)

The screenshot shows the 'Edit properties' dialog box for the Validation activity. The 'Properties' tab is selected. The 'Validation type' is set to 'Date time'. There are six input fields for date components: Year, Month, Day, Hour, Minute, and Second. Each input field has a small vertical ellipsis button to its right. The 'Return value' is set to 'IsValid'. The dialog has 'Save' and 'Close' buttons at the bottom right.

There are six validation types that covers different parts of a date time. The given values need to result in a valid date for the selected option.

The result is a boolean with value true if the given values are all falling in the valid ranges for the option. otherwise the boolean is false.

Date

Year, Month, Day.

Ranges:

Year= **1 - 9999**.

Month= **1 - 12**.

Day: **1 - N** (number of days in the specified month).

Example:

```
var IsValid: Boolean;  
var Year: Word := 2024;  
var Month: Word := 5;  
var Day: Word := 6;  
IsValid := IsValidDate(Year, Month, Day);
```

Result: True;

Date day

Year, day.

Ranges:

Year: **1 - 9999**.

Day: **1 - 365** (366 in leap years).

Example:

```
var IsValid: Boolean;  
var Year: Word := 2024;  
var Day: Word := 87;  
IsValid := IsValidDateDay(Year, Day);
```

Result: True;

Date month week

Year, Month, Week, Day.

Ranges:

Year: **1 - 9999**.

Month: **1 - 12**.

Week: **1 - N** (number of weeks in the specified month).

Day: **1 - 7**.

Example:

```
var IsValid: Boolean;  
var Year: Word := 2024;  
var Month: Word := 5;  
var WeekOfMonth: Word := 10;  
var DayOfWeek: Word := 6;  
IsValid := IsValidDateMonthWeek(Year, Month, WeekOfMonth,  
DayOfWeek);
```

Result: False;

Date time

Year, Month, Day, Hour, Minute, Second.

Ranges:

Year: **1 - 9999.**

Month: **1 - 12.**

Day: **1 - N** (number of days in the specified month).

Hour: **0-23, 24 is possible if minute and second are 0.**

Minute: **0-59.**

Second: 0-59.

Example:

```
var IsValid: Boolean;  
var Year: Word := 2024;  
var Month: Word := 5;  
var Day: Word := 6;  
var Hour: Word := 7;  
var Minute: Word := 2;  
var Second: Word := 80;  
IsValid := IsValidDateTime(Year, Month, Day, Hour, Minute,  
Second, MilliSecond);
```

Result: False;

Date week

Year, Week, Day

Ranges:

Year: **1 - 9999.**

Week: **1 - N** (number of weeks in the specified year).

Day: **1 - 7.**

Example:

```
var IsValid: Boolean;  
var Year: Word := 2024;  
var WeekOfYear: Word := 2;  
var DayOfWeek: Word := 6;  
IsValid := IsValidDateWeek(Year, WeekOfYear, DayOfWeek);
```

Result: True;

Time

Hour, Minute, Second

Ranges:

Hour: **0-23, 24 is possible if minute and second are 0.**

Minute: **0-59.**

Second: **0-59.**

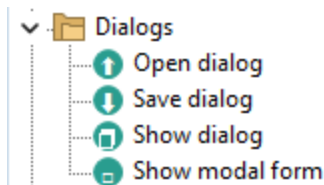
Example:





```
var IsValid: Boolean;  
var Hour: Word := 18;  
var Minute: Word := 5;  
var Second: Word := 5;  
var MilliSecond: Word := 0;  
IsValid := IsValidTime(Hour, Minute, Second, MilliSecond);
```

Result: True;

7.6 Dialogs

Dialogs are useful for showing information to a user. Note that these activities cannot be used in a service or back-end application, as displaying a dialog interrupts the flow of an application and waits for user input.



[OpenDialog](#)  95
[SaveDialog](#)  97
[ShowDialog](#)  98
[Show modal form](#)  99

7.6.1 OpenFileDialog

The open dialog file can be used to let the user select file(s) from the explorer.
uses [Vcl.Dialogs.TFileOpenDialog](#)

Edit properties

Properties | Information

Description

Default file path

Options

- ☐ Multiple files
- ☐ Path must exist
- ☐ File must exist
- ☐ Pick folders

Store selected file(s) in

ReturnList

Return value

IsExecuted

Save Close

Activity properties

```
begin
  var ReturnList: string;
  var IsExecuted: Boolean;
  var FileDialog := TFileOpenDialog.Create(nil);
  try
    IsExecuted := FileDialog.Execute;
    if IsExecuted then
      ReturnList := FileDialog.FileName;
    finally
      FileDialog.Free;
    end;
  end;
end;
```

Resulting code

The value of the 'store selected file(s) in' property is dependent on the option 'multiple files'. If 'multiple files' is selected, the property expects a string list to store the selected paths in. Otherwise a string variable is expected to store the selected path in.

If the 'Pick folders' option is selected, only folder can be selected.

The 'File/Path must exist' options are selectable for validation. If the options are not selected, the user is able to type in a file/folder and select it, regardless of it exists.

The 'Default file path' fills the selected file name for the user.

7.6.2 SaveDialog

The save file dialog can be used to let the user select the location of a file that needs to be stored/saved in the explorer.

uses [Vcl.Dialogs.TFileSaveDialog](#)

Activity properties

```
begin
  var ReturnList: string;
  var IsExecuted: Boolean;
  var FileDialog := TFileSaveDialog.Create(nil);
  try
    IsExecuted := FileDialog.Execute;
    if IsExecuted then
      ReturnList := FileDialog.FileName;
  finally
    FileDialog.Free;
  end;
end;
```

Resulting code

The value of the 'store selected file(s) in' property is dependent on the option 'multiple files'. If 'multiple files' is selected, the property expects a string list to store the selected paths in. Otherwise a string variable is expected to store the selected path in.

If the 'Pick folders' option is selected, only folder can be selected.

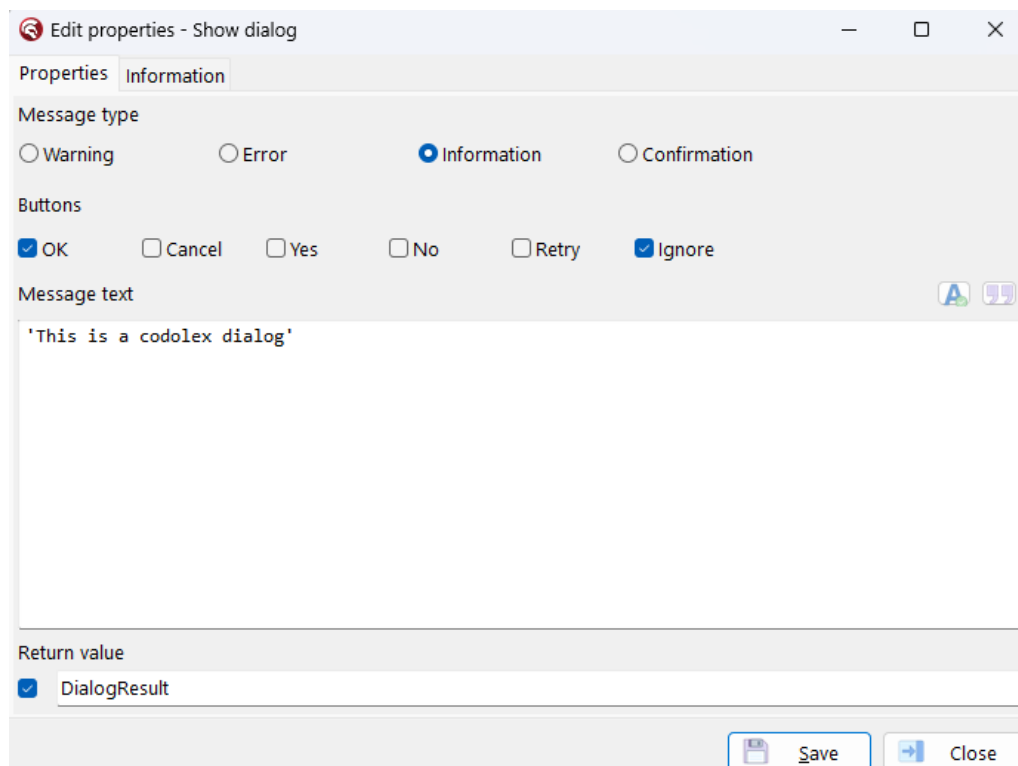
The 'File/Path must exist' options are selectable for validation. If the options are not selected, the user is able to type in a file/folder and select it, regardless of it exists.

The 'Default file path' fills the selected file name for the user.

7.6.3 ShowDialog

The **show dialog** activity can be used to show information to the user and/or to ask for user input.

uses [Vcl.Dialogs.MessageDlg](#)



Activity properties

```
begin
  var DialogResult: Integer;
  DialogResult := MessageDlg('This is a codolex dialog',
    TMsgDlgType.mtInformation, [TMsgDlgBtn.mbOK, TMsgDlgBtn.mbIgnore],
    0);
end;
```

Resulting code

The message text needs to be a string.

The message type defines the type of the dialog, 1 must be selected

The buttons provide options for the user, multiple options are possible. If no option is selected, the Ok button is shown. This defines the result integer options.

Ok -> 1
Cancel -> 2
Yes -> 6
No -> 7
Retry -> 4
Ignore -> 5

It's also possible to use the show dialog activity without 'Return value'. The result might not be needed in the case of just showing information. Turning of the return variable prevents hint's from the RAD Studio editor about unused variables.

7.6.4 Show modal form

Show modal form can be used to open any TCustomForm available in the project. uses [Vcl.Forms.TCustomForm.ShowModal](#)

The screenshot shows the 'Edit properties' dialog for a 'Show modal form' activity. The 'Properties' tab is selected. The 'Description' field is empty. The 'Namespace' field contains 'subfolder.vclformclass'. The 'Form class' field contains 'TFormClassName'. The 'Initialization code' field contains the comment '//code to initialize form'. The 'Return value' field contains 'ModalResult'. At the bottom right, there are 'Save' and 'Close' buttons.

Activity properties

begin

```
var ModalResult: Integer;  
var Form := TFormClassName.Create(nil);  
try  
    //code to initialize Form  
    ModalResult := Form.ShowModal;  
finally  
    Form.Free;  
end;  
end;
```

Resulting code

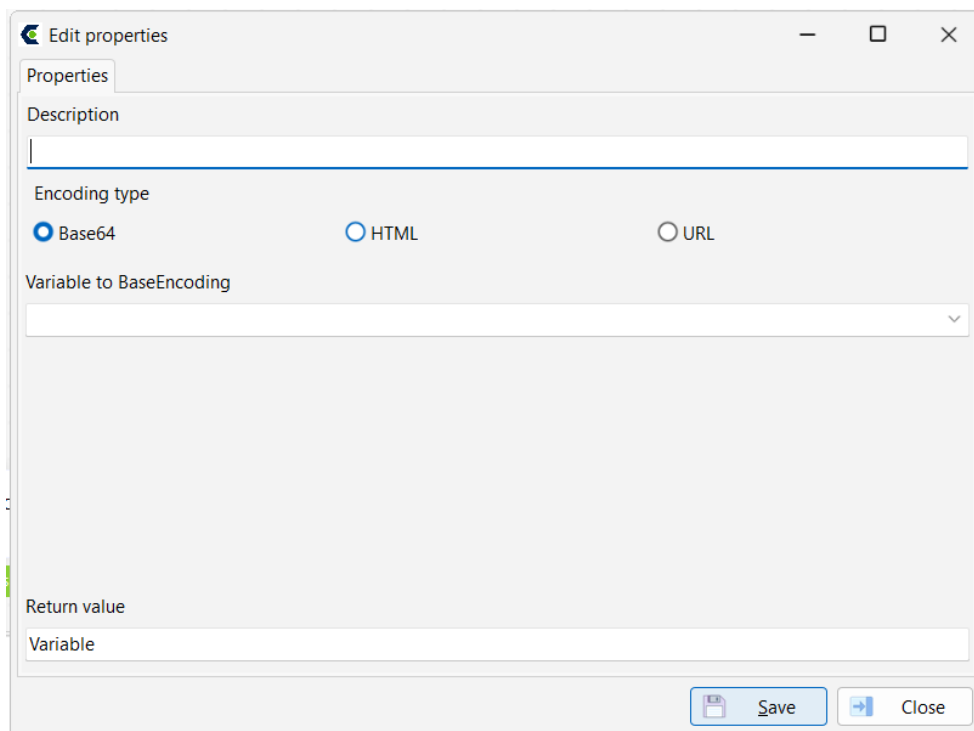
Namespace and folder class must be used to select the TCustomForm class that must be opened.

The initialization code will be added just before opening the form. The code can be used to call a function/procedure to set values of the class.
The initialized form can be referenced with 'Form'.

The modal return value will be stored in an integer when provided.

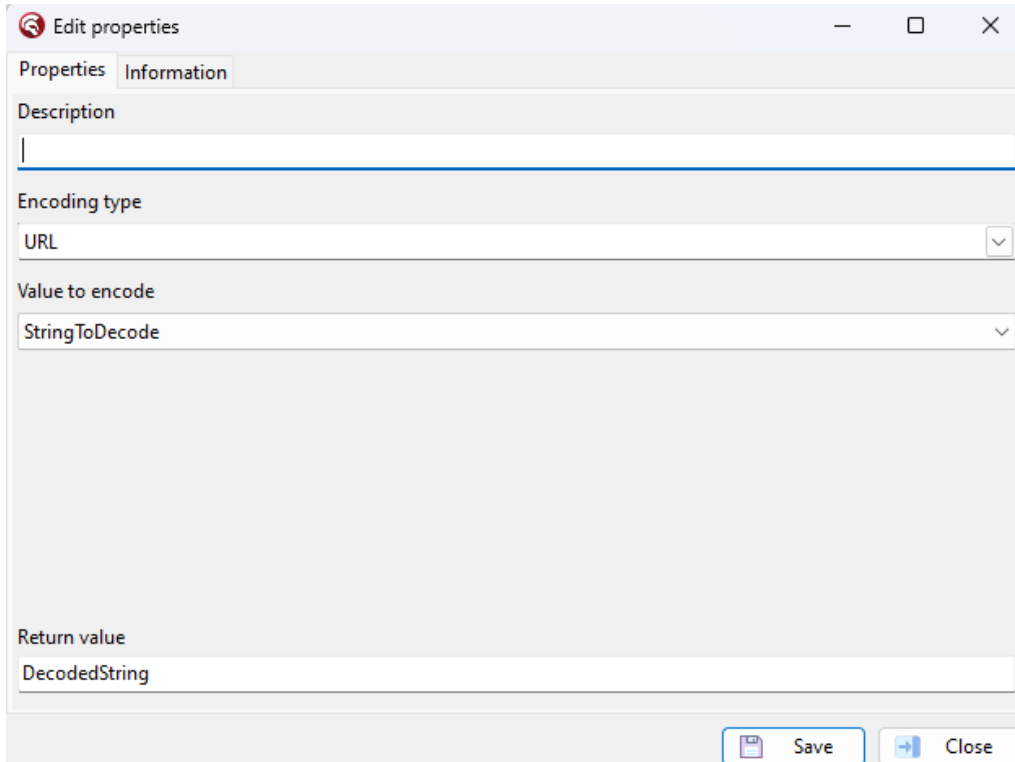
7.7 Encoding

Use the encoding options to decode and encode variables to URLs, Base64 or HTML Encoding code.



7.7.1 Decode

Decode a string with a decoding protocol
uses [System.NetEncoding.TNetEncoding.Decode](#)



Activity properties

```
begin
  var DecodedString: string;
  var Encoding := TURLEncoding.Create;
  try
    DecodedString := Encoding.Decode(StringToDecode);
  finally
    Encoding.Free;
  end;
end;
```

resulting code

Possible encoding types are:

- [URL](#)
- [HTML](#)
- [Base64](#)

7.7.2 Encode

Encode a string with an encoding protocol

uses [System.NetEncoding.TNetEncoding.Encode](#)

Edit properties

Properties | Information

Description

Encoding type
Base64

Value to encode
StringToEncode

Return value
EncodedString

Save Close

Activity properties

```
begin
  var EncodedString: string;
  var Encoding := {$IF CompilerVersion < 35.0}
  TBase64Encoding.Create{$ELSE}TBase64StringEncoding.Create{$ENDIF};
  try
    EncodedString := Encoding.Encode(StringToEncode);
  finally
    Encoding.Free;
  end;
end;
```

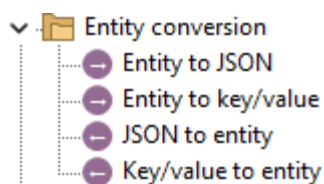
resulting code



Possible encoding types are:

- [URL](#)
- [HTML](#)
- [Base64](#)



7.8 Entity conversion

You can easily create Codolex entities using JSON or Key/value pair or convert Entities to these formats.



[Entity to JSON](#)  103
[JSON to entity](#)  104

This activity can be used to convert an entity to a JSON string or to a JSON object. The first return option in this activity is often used to pass an entity from Codolex to another service. To continue working with JSON in other Delphi code, a JSON object can be created. Please note that this instance is no longer managed by Codolex, and that this must be manually freed.

[Entity to key/value](#)  106
[Key/value to entity](#)  106

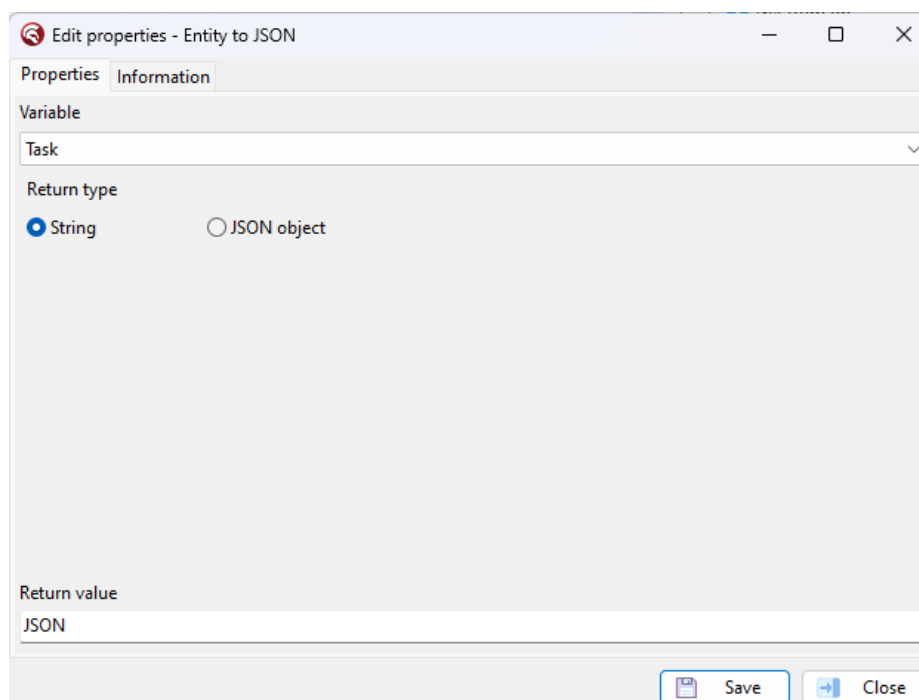
An entity can also be converted into a key/value pair. This makes it possible to store the key and value of an instance of an entity in a string or a TStringList. Note that as with the JSON object conversion, it is important to free the TStringList yourself. Codolex only returns the TStringList, and does not manage the TStringList.

The other two activities (JSON to entity and key/value to entity) are the opposite of those described above.

7.8.1 Entity to JSON

This activity can be used to convert an entity or list of entities to a JSON string or to a JSON object.

uses [System.JSON.TJSONObject](#)



Activity properties

```
begin
  var JSON: string;

  var Adapter:
    ICodolexEntityJSONAdapter<HelpAndManualScreenshots.DataSource.CodolexDataSource.ITask>;
  Adapter := TTaskJSONAdapter.Create;
  var JSONObject := Adapter.MapFromEntity(Task);
  try
    JSON := JSONObject.Format;
  finally
    JSONObject.Free;
  end;
end;
```

resulting code

The first return option in this activity is often used to pass an entity from Codolex to another service.

To continue working with JSON in other Delphi code, a JSON object can be created. Please note that this instance is no longer managed by Codolex, and that this must be manually freed.

7.8.2 JSON to entity

The JSON to entity activity can be used to parse a JSON string into an entity.

uses [System.JSON.TJSONObject](#)

Edit properties - JSON to entity

Properties Information

Expected results

☒ Single ☐ Multiple

Entity

CodolexDataSource.Task

Variable

StringWithJSON

Return value

TaskFromJSON

Save Close

Activity properties

```
begin
```

```

var TaskFromJSON:
HelpAndManualScreenshots.DataSource.CodoloxDatasource.ITask;
TaskFromJSON := nil;
var ResultCollection :=
TCodoloxCollections.CreateList<HelpAndManualScreenshots.DataSource.
CodoloxDatasource.ITask>;
var ParsedJson := TJsonObject.ParseJSONValue(StringWithJSON,
True, True);
try
var Entity:
HelpAndManualScreenshots.DataSource.CodoloxDatasource.ITask;
var Adapter:
ICodoloxEntityJSONAdapter<HelpAndManualScreenshots.DataSource.Codol
exDatasource.ITask>;
Adapter := TTaskJSONAdapter.Create;

if (ParsedJson is TJsonArray) then
begin
var JsonArray := ParsedJson as TJsonArray;
var NbOfItems := JsonArray.Count;

if (NbOfItems = 0) then
Exit;

for var ArrayElement in JsonArray do
begin
if not (ArrayElement is TJsonObject) then
Continue;

var ElementObject := ArrayElement.GetValue<TJsonObject>();

Entity := Adapter.MapToEntity(ElementObject);

ResultCollection.Add(Entity);
end;
end
else if ParsedJson is TJsonObject then
begin
Entity := Adapter.MapToEntity(ParsedJson);

ResultCollection.Add(Entity);
end;

finally
ParsedJson.Free;
TaskFromJSON := ResultCollection.First;
end;
end;

```

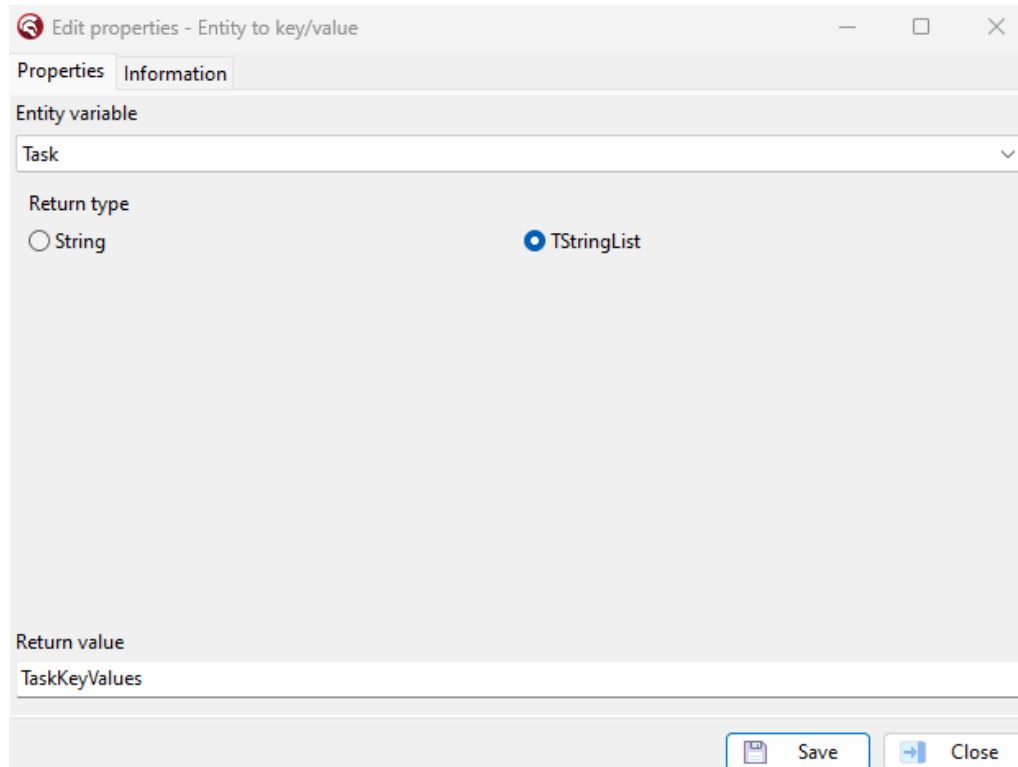
resulting code

When the option multiple for expected results is selected, the activity does return a list instead of a single entity.

7.8.3 Entity to key/value

The entity to key/value activity converts an entity into a string list of key/value pairs.

uses [System.Classes.TStringList](#)



Activity properties

```
begin
  var TaskKeyValues: TStringList;
  var StringList := TStringList.Create;
  StringList.AddPair('Description', Task.Description);
  StringList.AddPair('Done', Task.Done.Value.ToString);
  StringList.AddPair('DueDate', Task.DueDate);
  StringList.AddPair('Priority', Task.Priority);
  StringList.AddPair('TaskID', Task.TaskID.Value.ToString);
  TaskKeyValues := StringList;
end;
```

resulting code

The result can also be parsed into a string directly with the option return type string.

7.8.4 Key/value to entity

The key/value to entity activity converts a TStringList or a string of key/value pairs to an entity.

uses [System.Classes.TStringList](#)

Edit properties - Key/value to entity

Properties Information

Variable to convert

KeyValueTask

Entity

CodoloxDatasource.Task

Return value

NewTask

Save Close

Activity properties

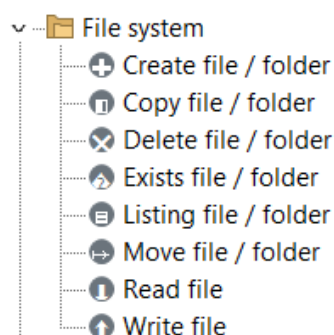
```
begin
  var TaskKeyValues: TStringList;
  var StringList := TStringList.Create;
  StringList.AddPair('Description', Task.Description);
  StringList.AddPair('Done', Task.Done.Value.ToString);
  StringList.AddPair('DueDate', Task.DueDate);
  StringList.AddPair('Priority', Task.Priority);
  StringList.AddPair('TaskID', Task.TaskID.Value.ToString);
  TaskKeyValues := StringList;
end;
```

resulting code

Both an string and TStringList are valid options for the variable to convert.

7.9 File system

The file system activities are all about reading and writing to the file system.



[Copy file/folder](#) ¹⁰⁸
[Create file/folder](#) ¹⁰⁹
[Delete file/folder](#) ¹¹¹
[Exists file/folder](#) ¹¹¹
[Get path part](#) ¹¹²
[Get system path](#) ¹¹⁵
[Listing file/folder](#) ¹¹⁶
[Move file/folder](#) ¹¹⁶
[Path validations](#) ¹¹⁸
[Read file](#) ¹²³
[Write file](#) ¹²³

7.9.1 Copy file/folder

The copy/folder file activity helps with duplicating resources in the system folders from one place to another.



uses [System.IOUtils.TDirectory.Copy](#)

uses [System.IOUtils.TFile.Copy](#)



Properties Information

Part

☒ Folder ☐ File

Source path  



'C:\...\Codolex\TestData'

Destination path  

'C:\...\Codolex\TestDataCopy'

Options

☐ Do when folder exists

 Save  Close

Activity properties

```
begin
  if TDirectory.Exists('C:\...\Codolex\TestData') then
    TDirectory.Move('C:\...\Codolex\TestData', 'C:\...
\Codolex\TestDataMoved');
end;
```

Resulting code

By default the folder option is selected. When a folder is copied, all nested files and folders are also copied.

To move a single file, use the file option.

Source and destination path must be provided, and must be valid folder or file paths, otherwise the generated code will result in an error.

Folder

The option 'do when folder exists' provides the ability to check if the source folder exists before copying the folder.

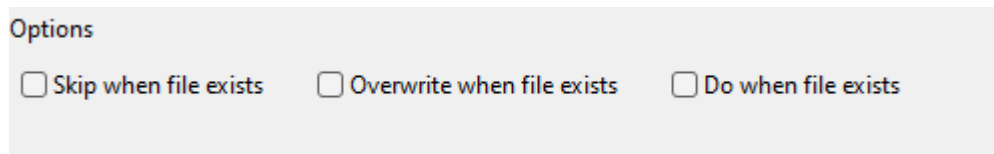
If the destination folder does not exist, a folder will be created.

If the destination folder does exist, the nested files and folder will be copied into the existing folder.

Files that are already present will not be overwritten.

File

For files, there are some extra options.



Options

☐ Skip when file exists ☐ Overwrite when file exists ☒ Do when file exists

The option 'do when file exists' provides the ability to check if the source file exists before copying the file.

The other options must be used exclusively and determines if the file must be overwritten or skipped if it already exists.

7.9.2 Create file/folder

The create file/folder activity can be used to create files or folder on the local system

uses [System.IOUtils.TDirectory.CreateDirectory](#)

uses [System.IOUtils.TFile.AppendAllText](#)

Properties Information

Part

☒ Folder ☐ File

Path

'C:\...\Codollex\TestData'

Options

☐ Skip when folder exists

Save Close

Activity properties

```
begin
  TDirectory.Copy('C:\...\Codollex\TestData', 'C:\...
\Codollex\TestDataCopy');
end;
```

Resulting code

Folder

The option 'do when folder exists' provides the ability to check if there is already a folder in place with the same name.

If this option is not set, and the folder is in place, the content of the folder will not be deleted.

File

For files, there are some other options.

Options

☐ Skip when file exists ☐ Overwrite when file exists

The options must be used exclusively and determines if the file must be overwritten or skipped if it already exists.

If the file must be overwritten, an empty file is created and the old file is deleted.

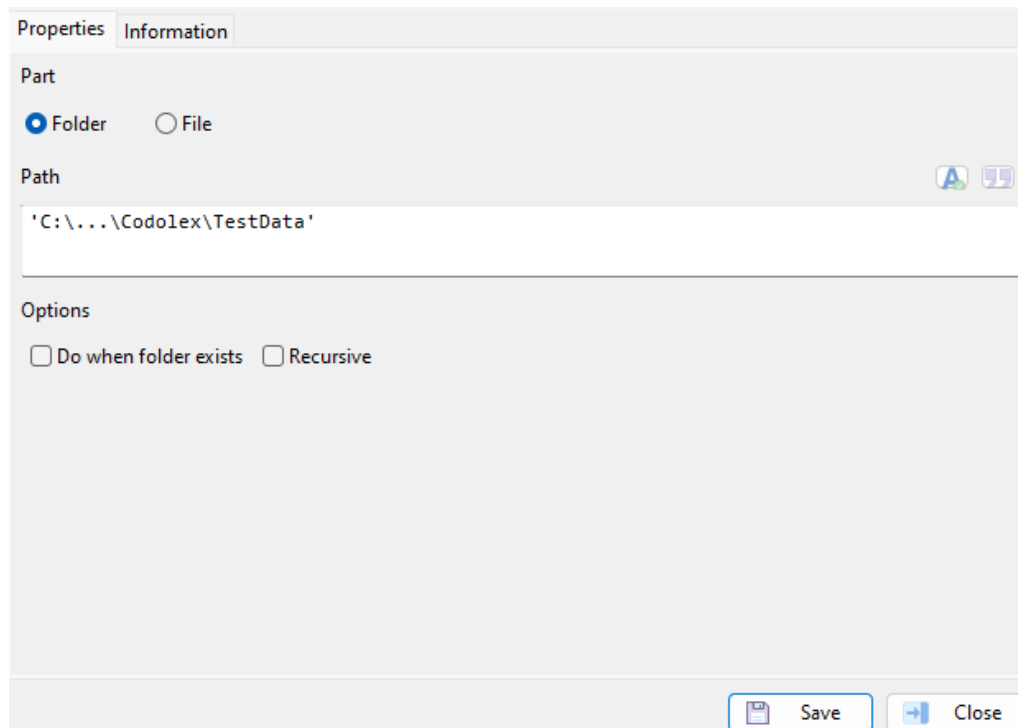
If Skip and Overwrite are both not selected, the file will not be created/overwritten when the file does exist.

7.9.3 Delete file/folder

The delete file folder activity helps with deleting files or folders from the local system.

Uses [System.IOUtils.TDirectory.Delete](#)

Uses [System.IOUtils.TFile.Delete](#)



ActivityProperties

```
begin
  TDirectory.Delete('C:\...\Codolex\TestData');
end;
```

Resulting code

Folder

The 'Do when folder exists' option provides a way to check if the folder exists before trying to delete it.

When the recursive option is not checked, and the folder contains files or folders, an [EInOutError](#)¹⁰¹ error will be thrown.

File

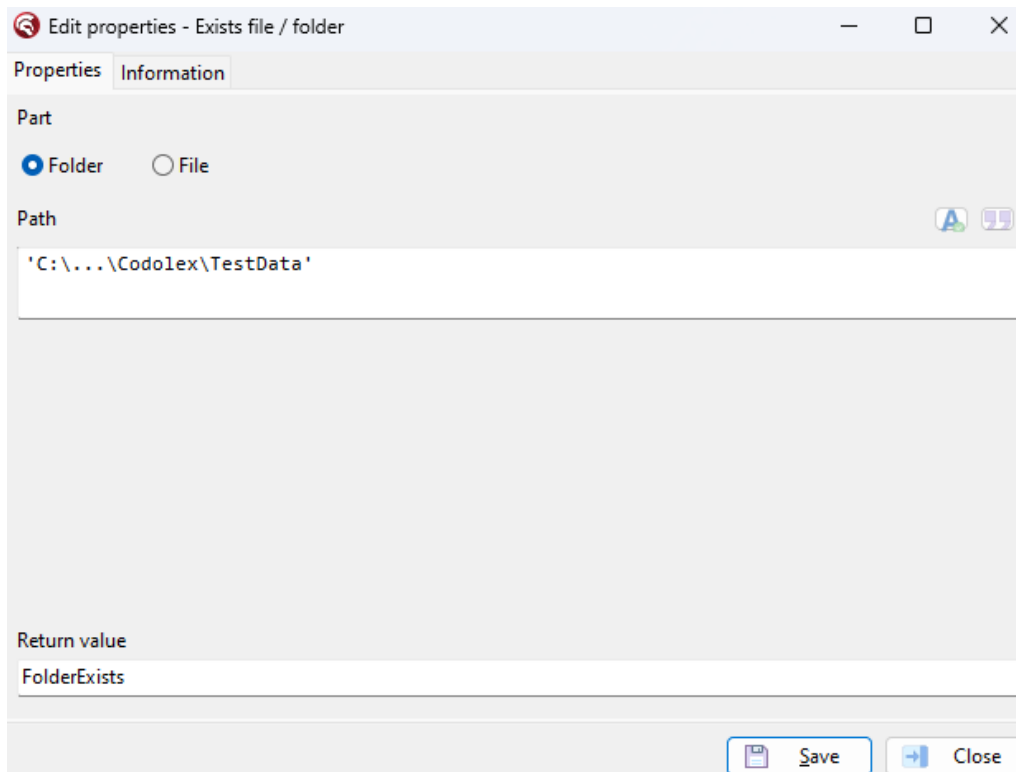
The 'Do when file exists' option provides a way to check if the file exists before trying to delete it.

7.9.4 Exists file/folder

The exists file/folder activity returns a boolean if the given path contains a file or folder.

uses [System.IOUtils.TDirectory.Exists](#)

uses [System.IOUtils.TFile.Exists](#)



Activity properties

```
begin
  TDirectory.Delete('C:\...\Codolox\TestData');
end;
```

resulting code

The activity id is available for both file and folder. However, if a folder is found when checking for a file (or the other way around), the result will be false.

7.9.5 Get path part

Get path part can be useful for getting different parts of a path you have. If you want to get the file name only from a path for example.

uses [System.IOUtils.TPath Methods](#)

Properties Information

Path part to return

File name

Path to use

FilePath

Return value

FileNameOfPath

Save Close

Activity properties

There are six different parts to retrieve from a path.

Directory name

Gets the directory name of the file path.

Example:

Filepath = 'C:\Users\Username\Documents\Codolex\Project.fcp'.

```
begin
  var PathPart: string;
  PathPart := TPath.GetDirectoryName(FilePath);
end;
```

Result = 'C:\Users\Username\Documents\Codolex'

File name

Gets the file name plus extension of the file path.

Example:

Filepath = 'C:\Users\Username\Documents\Codolex\Project.fcp'.

```
begin
  var PathPart: string;
  PathPart := TPath.GetFileName(FilePath);
end;
```

Result = 'Project.fcp'

File name without extension

Gets the file name without extension of the file path.

Example:

Filepath = 'C:\Users\Username\Documents\Codolex\Project.fcp'.

```
begin
  var PathPart: string;
  PathPart := TPath.GetFileNameWithoutExtension(FilePath);
end;
```

Result = 'Project'

Extension

Gets the extension of the file path.

Example:

Filepath = 'C:\Users\Username\Documents\Codolex\Project.fcp'.

```
begin
  var PathPart: string;
  PathPart := TPath.GetExtension(FilePath);
end;
```

Result = '.fcp'.

Full path

Gets the full path of the file path.

Example:

Filepath = 'C:\Users\Username\Documents\Codolex\Project.fcp'

```
begin
  var PathPart: string;
  PathPart := TPath.GetFullPath(FilePath);
end;
```

Result = 'C:\Users\Username\Documents\Codolex\Project.fcp'

Root path

Gets the root path of the file path.

Example:

Filepath = 'C:\Users\Username\Documents\Codolex\Project.fcp'

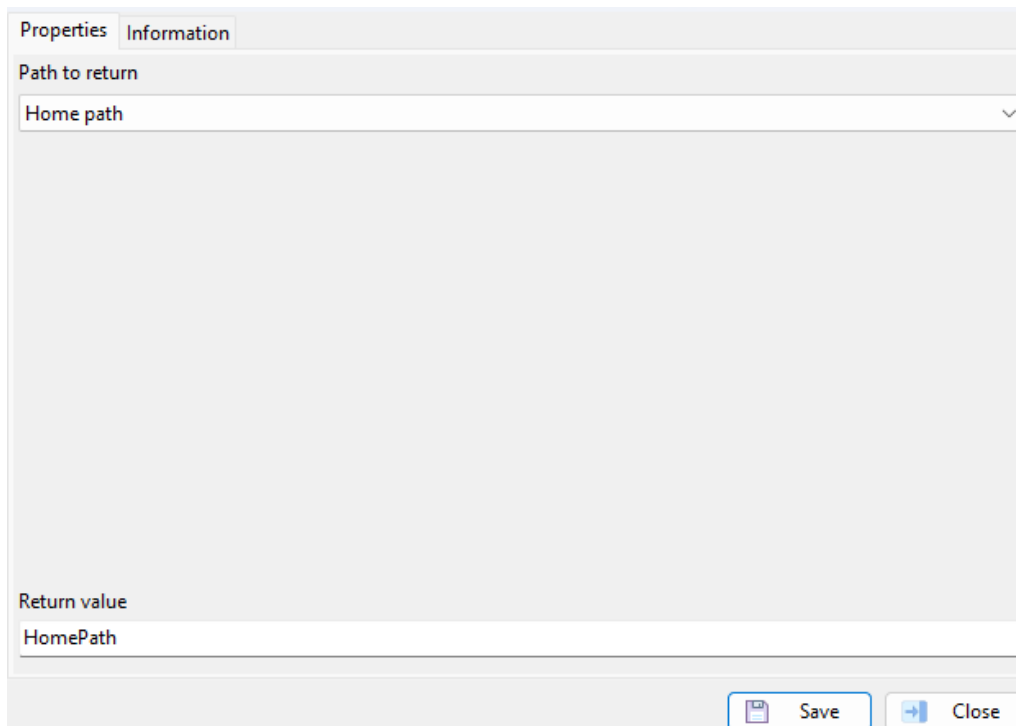
```
begin
  var PathPart: string;
  PathPart := TPath.GetPathRoot(FilePath);
end;
```

Result = 'C:\'

Providing an incorrect path will not result in errors, but may have unexpected outcomes.

7.9.6 Get system path

Get system path helps with getting the path for default folders on the system uses [System.IOUtils.TPath Methods](#)



activity properties

```
begin
  var HomePath: string;
  HomePath := TPath.GetHomePath;
end;
```

resulting code

Default paths to get:

[Alarm path](#) / [Shared alarm path](#)

[Camera path](#) / [Shared camera path](#)

[Cache path](#)

[Downloads path](#) / [Shared downloads path](#)

[Documents path](#) / [Shared documents path](#)

[Home path](#)

[Library path](#)

[Movies path](#) / [Shared movies path](#)

[Picture path](#) / [Shared picture path](#)

[Public path](#)

[Ringtone path](#) / [Shared ringtone path](#)
[Temp path](#)

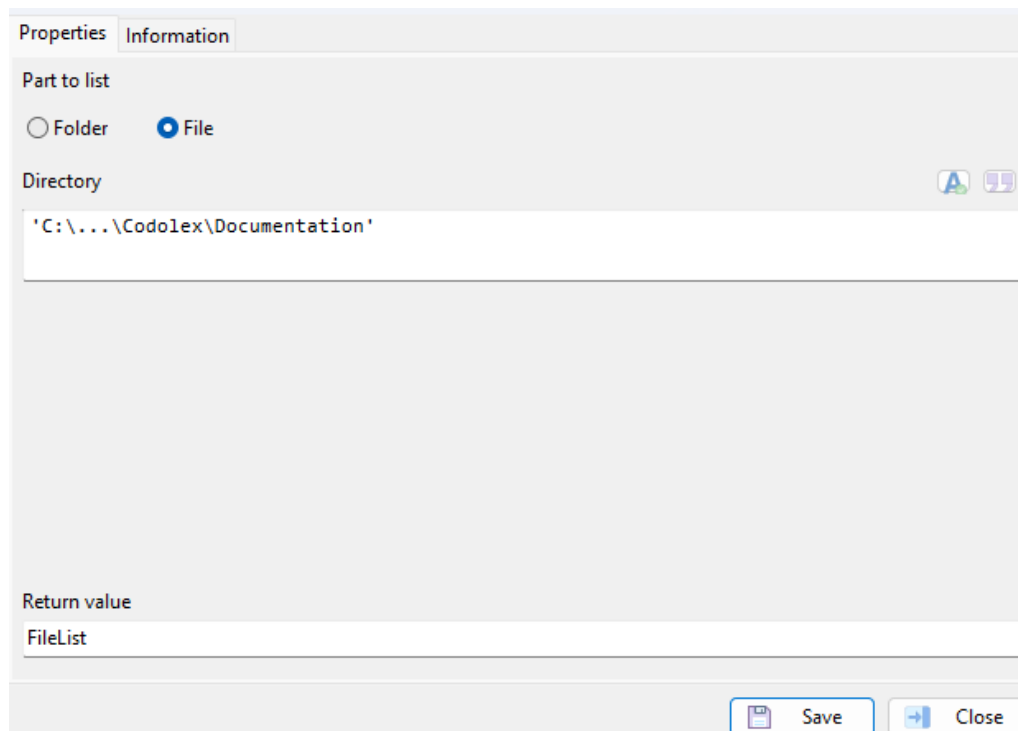
Some options have the possibility to get a shared path. This offers the choice between the path of the user, or the default path of the system

7.9.7 Listing file/folder

The listing file/folder activity list all the files or folder in a directory

uses [System.IOUtils.TDirectory.GetDirectories](#)

uses [System.IOUtils.TDirectory.GetFiles](#)



Activity properties

```
begin
  var FileList: ICodolexList<string>;
  FileList := TCodolexCollections.CreateList<string>;
  FileList.AddRange(TDirectory.GetFiles('C:\...\Codolex\Documentation'));
end;
```

Resulting code

When the option 'Folder' is selected, only folders will be listed.
When the option 'File' is selected, only files will be listed.

When a non-valid path is provided, a [EDirectoryNotFoundException](#) is thrown

7.9.8 Move file/folder

The move file/folder activity moves files and folders from one place to another on the local file system

Uses [System.IOUtils.TDirectory.Move](#)

Uses [System.IOUtils.TFile.Move](#)

Properties Information

Part

☒ Folder ☐ File

Source path

'C:\...\Codollex\TestData'

Destination path

'C:\...\Codollex\TestDataCopy'

Options

☒ Do when folder exists

Save Close

Activity properties

```
begin
  if TDirectory.Exists('C:\...\Codollex\TestData') then
    TDirectory.Move('C:\...\Codollex\TestData', 'C:\...
\Codollex\TestDataMoved');
end;
```

Resulting code

By default the folder option is selected. When a folder is moved, all nested files and folders are also moved. After moving, the old folder is deleted.

To move a single file, use the file option.

Source and destination path must be provided, and must be valid folder or file paths, otherwise the generated code will result in an error.

Folder

The option 'do when folder exists' provides the ability to check if the source folder exists before moving the folder.

If the destination folder does not exist, a folder will be created.

If the destination folder does exist, the nested files and folder will be moved into the existing folder.

Files that are already present will not be overwritten.

File

For files, there are some extra options.

Options

☐ Skip when file exists ☒ Do when file exists

The option 'do when file exists' provides the ability to check if the source file exists before copying the file.

The option 'Skip when file exists' determines if the file must be overwritten or skipped if it already exists.

7.9.9 Path validations

Path validations can be used to check on different parts if a path valid uses [System.IOUtils.TPath Methods](#)

Activity properties

```
begin
  var Variable: Boolean;
  try
    var FileNamePart := TPath.GetFileName(PathToValidate);
    var PathPart := TPath.GetDirectoryName(PathToValidate);

    var IsValidFileName :=
      TPath.IsValidFileNameChars(FileNamePart, False);
    var IsValidPathPart := TPath.IsValidPathChars(PathPart,
      False);

    Variable := IsValidPathPart and IsValidFileName;
  except
```

```
{ $IF CompilerVersion > 34.0 }  
on E: EInOutArgumentException do  
{ $ELSE }  
on E: EArgumentException do  
{ $ENDIF }  
    Variable := False;  
end;  
end;
```

Resulting code

The option "Use wild cards" will enable the wildcards (*) and (?) in the given path/filename value

Available methods

Matches pattern

Has an extra string input for pattern

Returns True if the given value matches the specified pattern.

Example:

Path to validate = 'Project2024.fcp'.

Pattern to match = 'Project*.fcp'

```
begin  
    var Variable: Boolean;  
    Variable := TPath.MatchesPattern(PathToValidate, 'Project*.fcp',  
    False);  
end;
```

Result = False

Drive exists

Checks whether the drive letter used in the given path actually exists.

Example:

Path to validate = 'C:\Users\Username\Documents\Codolox\NonProject.fcp'.

```
begin  
    var Variable: Boolean;  
    Variable := TPath.DriveExists(PathToValidate);  
end;
```

Result = False

[Has Extension](#)

Checks whether a given file name has an extension part.

Example:

Path to validate = 'C:\Users\Username\Documents\Codolex\Project.fcp'.

```
begin
  var Variable: Boolean;
  Variable := TPath.HasExtension(PathToValidate);
end;
```

Result = True

[Has valid filename chars](#)

Checks whether a given file name contains only allowed characters.

Example:

Path to validate = 'C:\Users\Username\Documents\Codolex\Project.fcp'.

```
begin
  var Variable: Boolean;
  Variable := TPath.HasValidFileNameChars(PathToValidate, False);
end;
```

Result = False

[Has valid path chars](#)

Example:

Path to validate = 'C:\Users\Username\Documents\Codolex\Project.fcp'.

```
begin
  var Variable: Boolean;
  Variable := TPath.HasValidPathChars(PathToValidate, False);
end;
```

Result = True

[Is drive rooted](#)

Checks whether a given path is absolute and starts with a drive letter.

Example:

Path to validate = 'C:\Users\Username\Documents\Codolex\Project.fcp'.

```
begin
  var Variable: Boolean;
  Variable := TPath.IsDriveRooted(PathToValidate);
end;
```

Result = True

Is extended prefix

Checks whether a given path has an extended prefix. Extended meaning longer than the max path length of 260 chars.

Example:

Path to validate = 'C:\Users\Username\Documents\Codolx\Project.fcp'.

```
begin
  var Variable: Boolean;
  Variable := TPath.IsExtendedPrefixed(PathToValidate);
end;
```

Result = True

Is absolute path

Checks whether a given path is an absolute path .

Example:

Path to validate = 'C:\Users\Username\Documents\Codolx\Project.fcp'.

```
begin
  var Variable: Boolean;
  Variable := TPath.IsPathRooted(PathToValidate);
end;
```

Result = True

Is relative path

Checks whether a given path is a relative path, in other words, not rooted to a drive.

Example:

Path to validate = 'C:\Users\Username\Documents\Codolx\Project.fcp'.

```
begin
  var Variable: Boolean;
  Variable := TPath.IsRelativePath(PathToValidate);
end;
```

Result = True

Is UNC path

Checks whether a given path is in UNC (Universal Naming Convention) format.

Example:

Path to validate = 'C:\Users\Username\Documents\Codolx\Project.fcp'.

```
begin
```

```
var Variable: Boolean;  
Variable := TPath.IsUNCPath(PathToValidate);  
end;
```

Result = False

[is UNC rooted](#)

Checks whether the given path is UNC-rooted, where UNC stands for Universal Naming Convention.

Example:

Path to validate = 'C:\Users\Username\Documents\Codolox\Project.fcp'.

```
begin  
var Variable: Boolean;  
Variable := TPath.IsUNCRooted(PathToValidate);  
end;
```

Result = False

[Is valid filename chars](#)

Checks whether a given character is allowed in a file name.

Example:

String to validate = 'C'.

```
begin  
var Variable: Boolean;  
Variable := TPath.HasValidFileNameChars(PathToValidate[1]);  
end;
```

Result = True

[Is valid path chars](#)

Checks whether a given character is allowed in a path string.

Example:

String to validate = 'C'.

```
begin  
var Variable: Boolean;  
Variable := TPath.HasValidPathChars(PathToValidate[1]);  
end;
```

Result = True

Is valid path

Checks whether a given value name contains only allowed characters for file and path.

Uses [Has valid filename chars](#) and [Has valid path chars](#)

Is valid and existing path

Checks whether a given value name contains only allowed characters for file and path, and if the path actually exists.

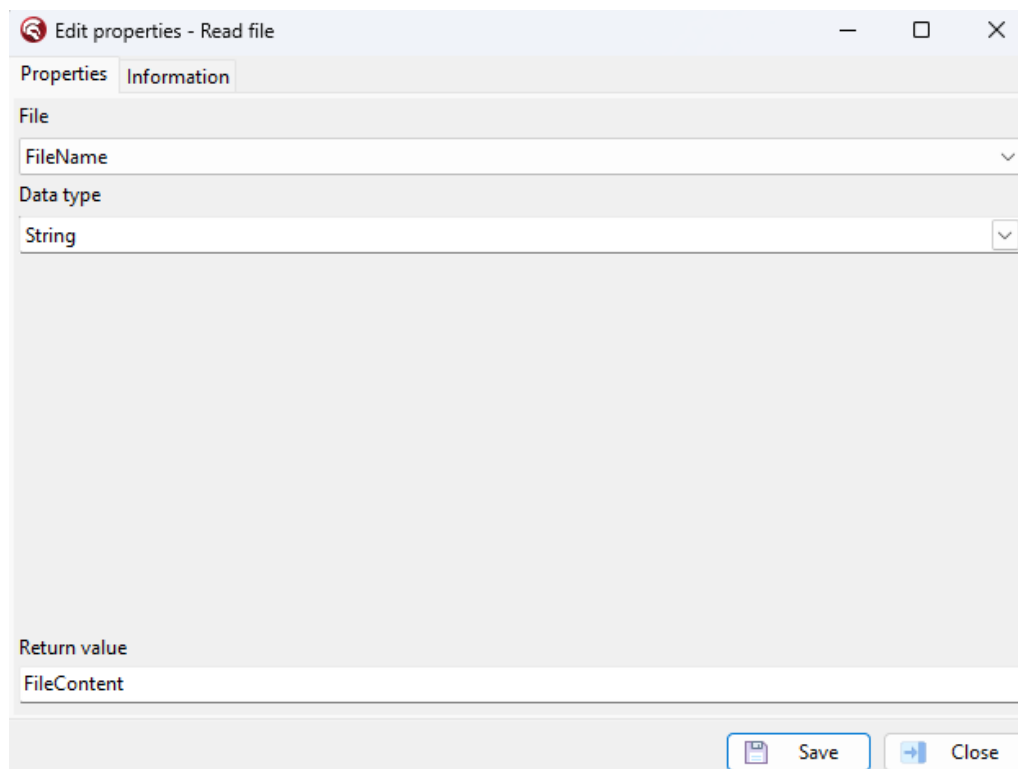
Uses [Has valid filename chars](#) and [Has valid path chars](#) plus [Directory exists](#) or [File exists](#)

7.9.10 Read file

The read file activity can be used to read content from a file

uses [System.IOUtils.TFile.ReadAllText](#)

uses [System.Classes.TStrings.LoadFromFile](#)



Activity properties

```
begin
  var FileContent: string;
  FileContent := TFile.ReadAllText(FileName);
end;
```

Resulting code

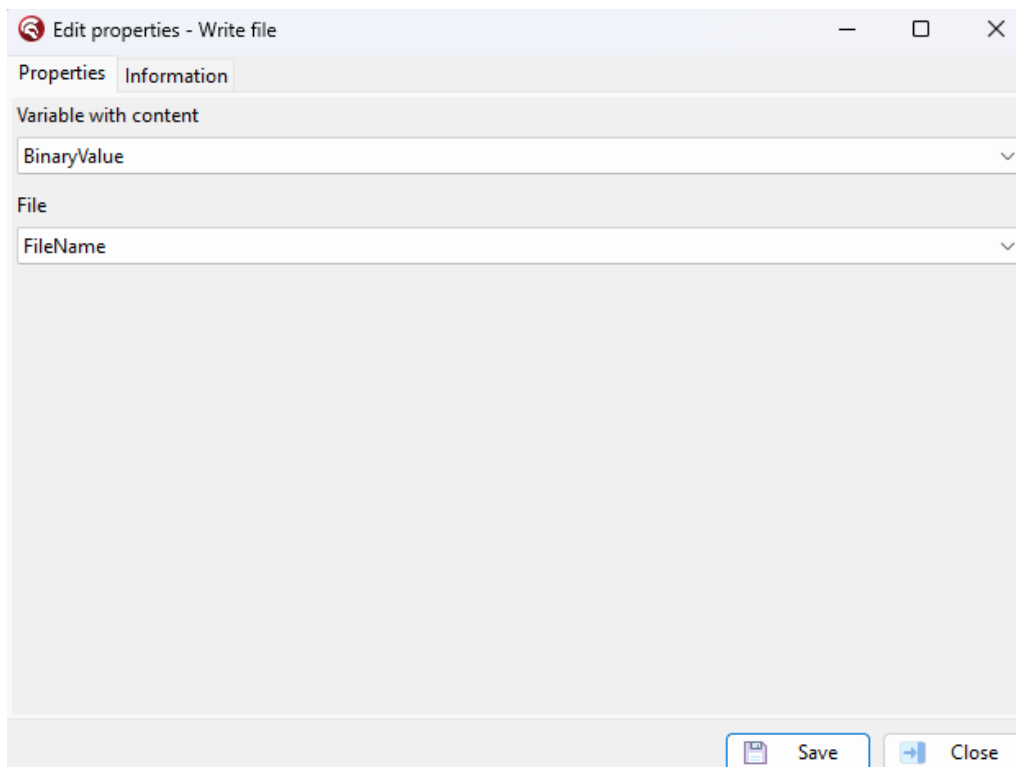
Depending on Data type, the return value is a string or binary.

7.9.11 Write file

The write file activity can be used to write or append data to a file

uses [System.IOUtils.TFile.WriteAllText](#) and [System.IOUtils.TFile.AppendAllText](#)

uses [System.Classes.TStrings.SaveToFile](#)



Activity properties

```
begin
  BinaryValue.Stream.Position := 0;
  BinaryValue.Stream.SaveToFile(FileName);
end;
```

Resulting code

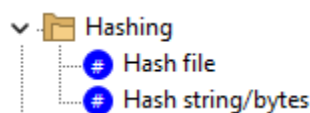
If there is no existing file at the given file path, a new file will be created.

It's possible to provide a binary or string value.

When using a string value, the option 'Append content' is also available. This determines if the string should be added at the end of the file, or if the existing file should be overridden.

7.10 Hashing

Hashing converts data into a fixed-size string of characters, typically for security and efficiency. It ensures data integrity, speeds up data retrieval, enables safe password storage by turning readable data into unreadable strings, and helps in quickly comparing large datasets by comparing smaller hashed values instead. Codollex supports MD5, SHA1 and SHA2 hashing algorithms.



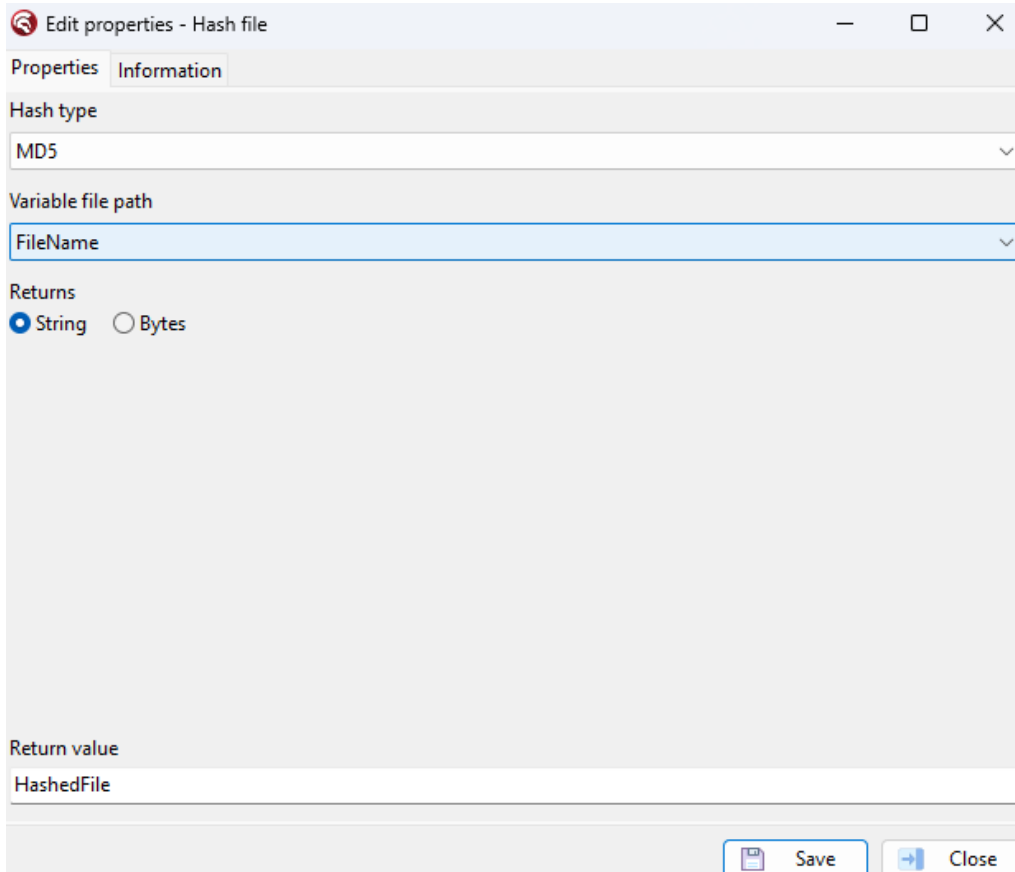
[Hash file](#) 125

[Hash string/bytes](#) 

7.10.1 Hash file

The hash file activity hashes the contents of a file and returns it as string or bytes array variable

uses [System.Hash](#)



Edit properties - Hash file

Properties Information

Hash type

MD5

Variable file path

FileName

Returns

☒ String ☐ Bytes

Return value

HashedFile

Save Close

Activity properties

```
begin
  var HashedFile: string;
  HashedFile := THashMD5.GetHashStringFromFile(FileName);
end;
```

Resulting code

Supported hashing types

- MD5
- SHA1
- SHA2_224
- SHA2_256
- SHA2_384
- SHA2_512
- SHA2_512_224
- SHA2_512_256

7.10.2 Hash string/bytes

The hash file activity hashes the contents of a string variable and returns it as string or bytes array variable

uses [System.Hash](#)

Edit properties - Hash string/bytes

Properties Information

Hash type

MD5

Variable

Content

Returns

☐ String ☒ Bytes

Return value

Variable

Save Close

Activity properties

```
begin
  var Variable: TBytes;
  Variable := THashMD5.GetHashBytes(Content);
end;
```

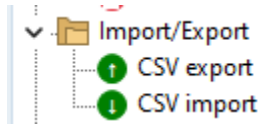
Resulting code

Supported hashing types

- MD5
- SHA1
- SHA2_224
- SHA2_256
- SHA2_384
- SHA2_521
- SHA2_521_224
- SHA2_512_256

7.11 Import/Export

The Import/Export section contains activities for importing and exporting data



[CSV export](#)¹²⁷

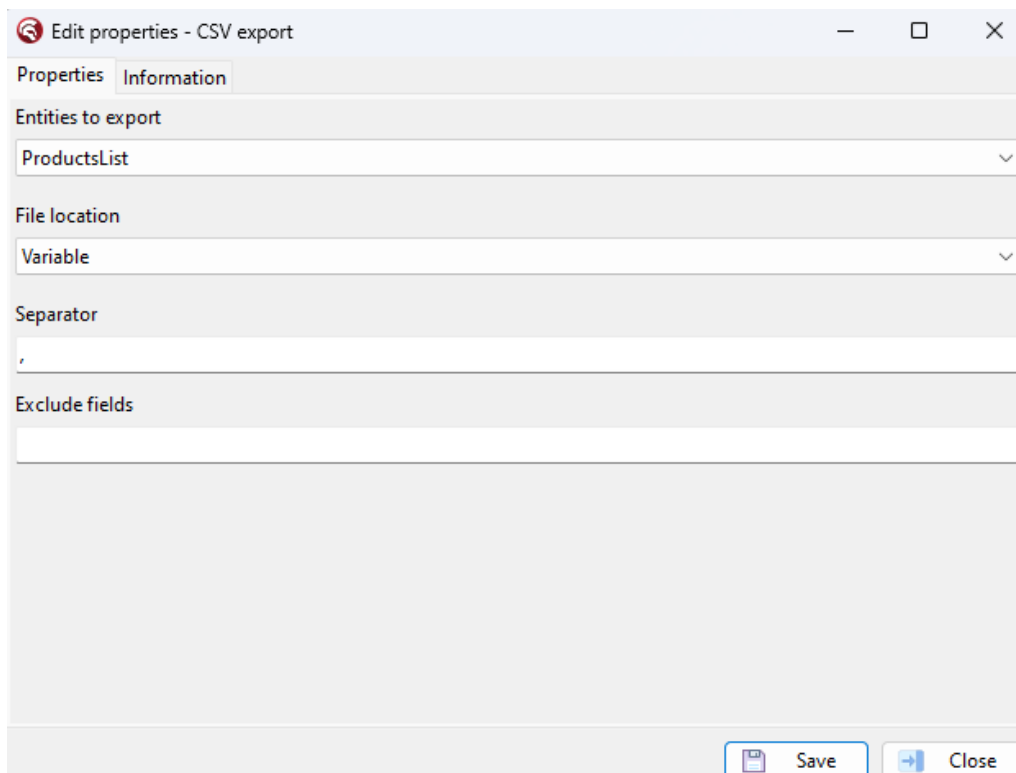
[CSV import](#)¹²⁸

7.11.1 CSV export

The CSV export activity can be used to export a list of entities to a CSV file.

Uses [System.Classes.TStringList](#)

Uses [System.Classes.TStrings.SaveToFile](#)



Activity properties

```
begin
var CSVList := TStringList.Create;
try
    CSVList.Add('ProductID,ProductName,SupplierID,CategoryID,QuantityPerUnit,UnitPrice,UnitsInStock,UnitsOnOrder,ReorderLevel,Discontinued');
    for var Entity in ProductsList do
    begin
        var CSVValue: string;
```

```
        CSVValue := string.Join(',', [Entity.ProductID.Value,  
        Entity.ProductName.Value, Entity.SupplierID.Value,  
        Entity.CategoryID.Value, Entity.QuantityPerUnit.Value,  
        Entity.UnitPrice.Value, Entity.UnitsInStock.Value,  
        Entity.UnitsOnOrder.Value, Entity.ReorderLevel.Value,  
        Entity.Discontinued.Value]);  
        CSVList.Add(CSVValue);  
    end;  
    CSVList.SaveToFile(ExportFileName);  
finally  
    CSVList.Free;  
end;  
end;
```

Resulting code

The separator value is a ',' by default, but can be changed to other separators like ';'.

The associations of entities are not included in the export, only the field for association id is included if there is one

Known issue

Exclude fields is not implemented in the current version

7.11.2 CSV import

The CSV Import activity can be used to import entities directly from a csv file

Uses [System.Classes.TStringList](#)

Uses [System.IOUtils.TFile.ReadAllLines](#)

Edit properties - CSV import

Properties Information

Entity to import
Codolex.Products

Path to importfile
ImportFileName

Separator
,

Return value
ProductsList

Save Close

Activity properties

```

var ProductsList:
ICodolexList<HelpAndManualScreenshots.DataSource.Codolex.IProducts>
;
var CSVLines := TFile.ReadAllLines(ImportFileName);

var ColumnList := TStringList.Create;
try
    ColumnList.Delimiter := ',';
    ColumnList.StrictDelimiter := True;
    ColumnList.DelimitedText := CSVLines[0];
    for var ColumnIndex := 0 to ColumnList.Count -1 do
        ColumnList[ColumnIndex] := Trim(ColumnList[ColumnIndex]);

    ProductsList :=
TCodolexList<HelpAndManualScreenshots.DataSource.Codolex.IProducts>
.Create;

    var IsFirstLine := True;
    for var Line in CSVLines do
    begin
        if IsFirstLine then
        begin
            IsFirstLine := False;
            Continue;
        end;

        var Values := Line.Split([';']);
        var CSVEntity:
HelpAndManualScreenshots.DataSource.Codolex.IProducts;

```

```

    CSVEntity :=
HelpAndManualScreenshots.DataSource.Codolx.TProducts.Create;
    var FieldCount := Length(Values);

    var FieldPos := ColumnList.IndexOf('ProductID');
    if (FieldPos > -1) and (FieldPos < FieldCount) then
        CSVEntity.ProductID := Values[FieldPos].ToInteger;

    ...

    FieldPos := ColumnList.IndexOf('Discontinued');
    if (FieldPos > -1) and (FieldPos < FieldCount) then
        CSVEntity.Discontinued := Values[FieldPos];

    ProductsList.Add(CSVEntity);
end;
finally
    ColumnList.Free;
end;

```

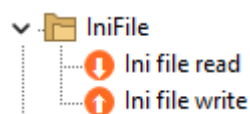
Resulting code

The separator value is a ',' by default, but can be changed to other separators like ';'.

The associations of entities are not included in the import, the association can be set after an import with id value.

7.12 IniFile

To use ini files, it is necessary to have an entity corresponding to the ini file structure. In the Ini file read activity, default values can be specified if the fields do not occur in the ini file.



[IniFile Datasource](#) 51

[IniFile Read](#) 130

[IniFile Write](#) 131

7.12.1 IniFile Read

The IniFile Read activity can be used to get data from a local ini file into a variable.
uses [System.IniFiles.TIniFile](#)

Edit properties - Ini file read

Properties Information

Variable for INI values

keys

Path to INI file

IniFilePath

Default values

Name	Type	Value
APIKey1	String	'123'
APIKey2	String	

Save Close

Activity properties

```
var IniFile := TIniFile.Create(IniFilePath);
try
  keys.APIKey1 := IniFile.ReadString('keys', 'APIKey1', '123');
  keys.APIKey2 := IniFile.ReadString('keys', 'APIKey2', '');
finally
  IniFile.Free;
end;
```

Resulting code

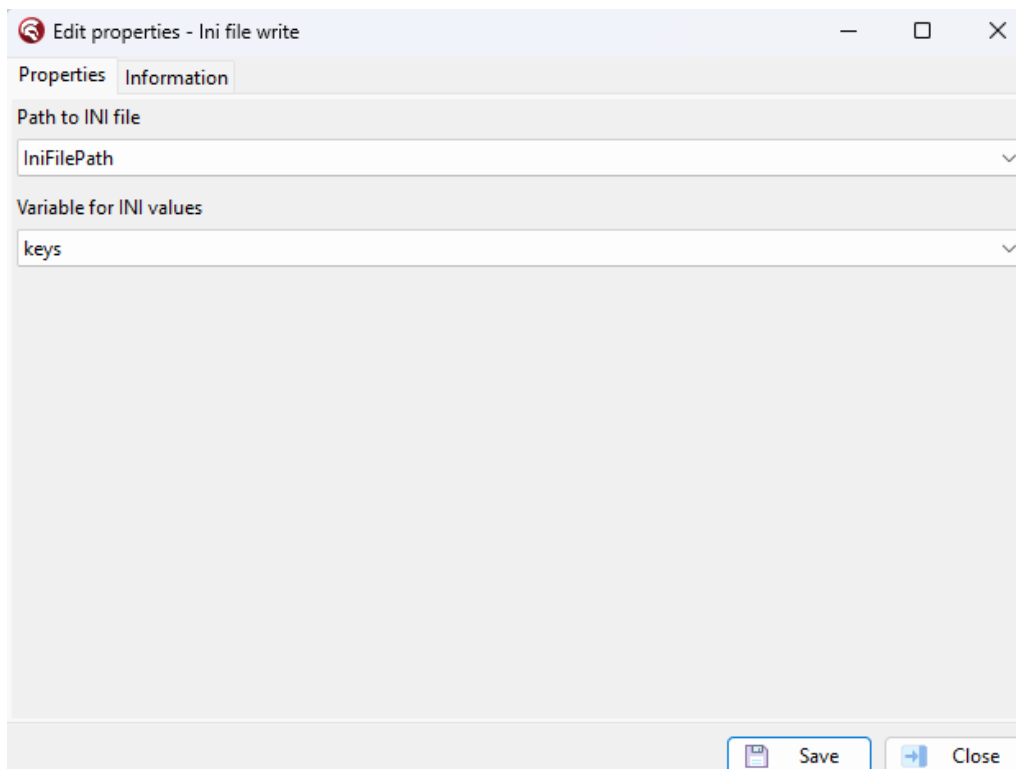
The activity expects a path to the ini file, and variable to store the values in.

It's recommended to create an inifile datasource to get the variable to store the ini values in.

More information about the datasource: [IniFile Datasource](#)

7.12.2 IniFile Write

The IniFile Write activity can be used to store data in a local ini file from a variable uses [System.IniFiles.TIniFile](#)



Activity properties

```
var IniFile := TIniFile.Create(IniFilePath);
try
  IniFile.WriteString('keys', 'APIKey1', keys.APIKey1);
  IniFile.WriteString('keys', 'APIKey2', keys.APIKey2);
finally
  IniFile.Free;
end;
```

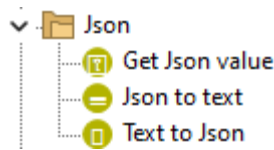
Resulting code

The activity expects a path to the ini file, and variable with values to write from. Keep in mind that all values get updated, even if a value is empty.

It's recommended to create an inifile datasource to get the variable to write from. More information about the datasource: [IniFile Datasource](#)⁵¹

7.13 JSON

The JSON activities are available to handle objects of TJSONValue type. This allows you to work with the JSON objects directly instead of handling the json as raw string. This could be helpfull for traversing the data before importing it into entities, or adding data to the JSON after it's been parsed from an entity.



[TextToJSON](#) ¹³³

[JSON to text](#) ¹³⁴

[Get JSON value](#) ¹³⁴

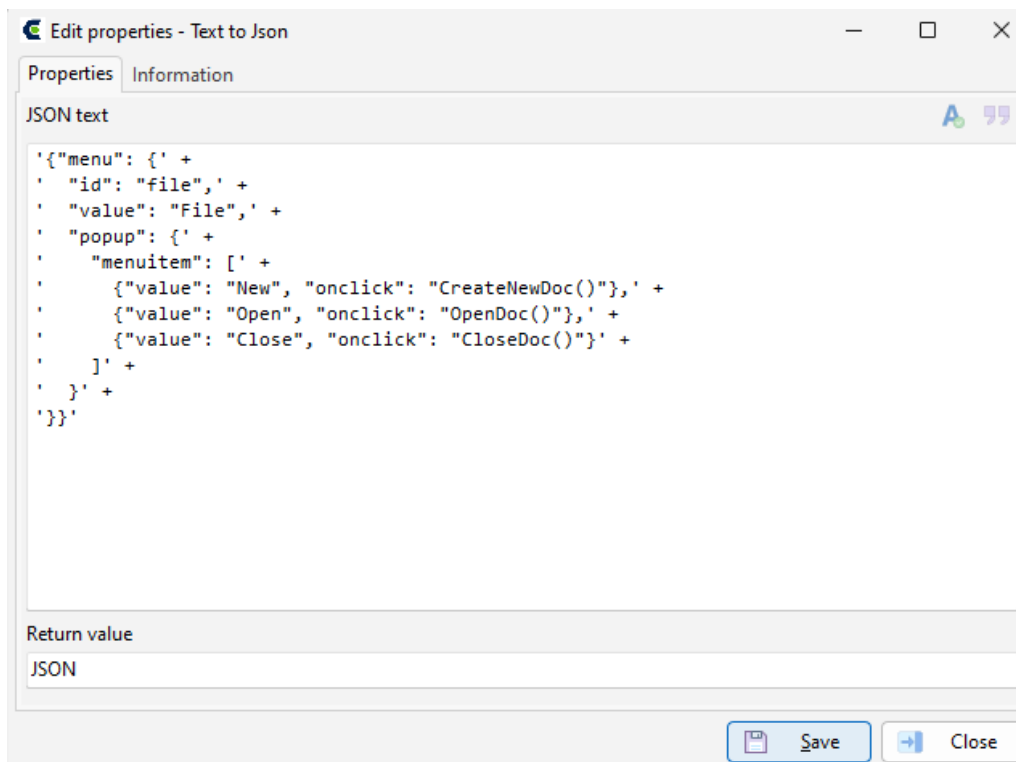
For more information about JSON

[JSON datasource](#) ⁵²

[Entity conversion](#) ¹⁰²

7.13.1 Text to JSON

The text to JSON Activity parses a string into a TJSONValue object.
uses [System.JSON.TJSONObject.ParseJSONValue](#)



Activity properties

```

begin
  var JSON: TJSONValue;
  JSON := TJsonObject.ParseJsonValue('{ "menu": { ' +
  '  "id": "file",' +
  '  "value": "File",' +
  '  "popup": { ' +
  '    "menuitem": [ ' +
  '      { "value": "New", "onclick": "CreateNewDoc()" }, ' +
  '      { "value": "Open", "onclick": "OpenDoc()" }, ' +
  '      { "value": "Close", "onclick": "CloseDoc()" } ' +
  '    ] ' +
  '  } ' +
  '}'
  
```

```
'    ]' +  
'  }' +  
'}}');  
end;
```

Resulting code

7.13.2 JSON to text

Use the JSON to Text activity to format a JSONValue to a string
uses [System.JSON.TJSONValue](#)



Edit properties - Json to text

Properties Information

Variable with JSON object

JSON

Return value

JSONText

Save Close

Activity properties

```
begin  
  var JSONText: string;  
  JSONText := JSON.Format;  
end;
```

Resulting Code

7.13.3 Get JSON value

The get JSON Value activity can be used to get a variable from an existing JSON Value.
uses [System.JSON.TJSONValue](#)

Edit properties - Get Json value

Properties Information

Variable with JSON object

JSON

Key name with value

'menu'

Data type

JSONValue

Return value

MenuJSON

Save Close

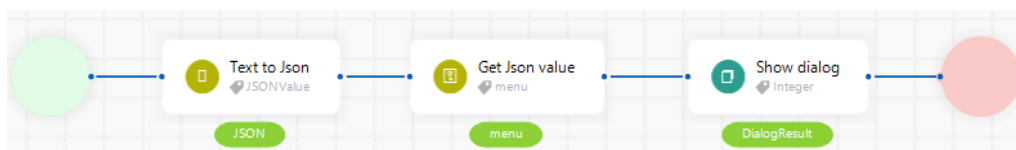
Activity properties

```
begin
  var MenuJSON: TJSONValue;
  MenuJSON := JSON.GetValue<TJSONValue>('menu');
end;
```

Resulting code

The 'Key name with value' must be a attribute of the provided JSON Value.
The data type could be any data type if the value of the attribute can be parsed into an object.

e.g.

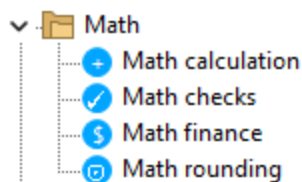


```
begin
  var JSON: TJSONValue;
  JSON := TJsonObject.ParseJsonValue('{ "menu": {' +
  '  "id": "other",' +
  '  "value": "File",' +
  '  "popup": {' +
  '    "menuItem": [' +
  '      {"value": "New", "onclick": "CreateNewDoc()"},' +
  '      {"value": "Open", "onclick": "OpenDoc()"},' +
  '      {"value": "Close", "onclick": "CloseDoc()"}' +
  '    ]' +
  '  }' +
  '});
```

```
var menu:
HelpAndManualScreenshots.DataSource.JSONMenuDataSource.Imenu;
var JsonAdapter:
ICodolexEntityJSONAdapter<HelpAndManualScreenshots.DataSource.JSONM
enuDataSource.Imenu>;
    JsonAdapter := TmenuJSONAdapter.Create;
var KeyNameValue := JSON.GetValue<TJsonValue>('menu');
var JsonValue := JsonAdapter.MapToEntity(KeyNameValue);
menu := JsonValue;
var DialogResult: Integer;
    DialogResult := MessageDlg(menu.id, TMsgDlgType.mtInformation,
[ TMsgDlgBtn.mbOK], 0);
end;
```

7.14 Math

the Math activities are specific activities that could help with common calculations.



[Calculation](#) 136
[Checks](#) 140
[Finance](#) 143
[Rounding](#) 148

7.14.1 Calculation

The calculation activity contains a set of math functions.

Uses [System](#)

Uses [System.Math](#)

Edit properties - Math calculation

Properties Information

Calculation type

Absolute value

Number value

varDecimal

Return value

Calculation

Save Close

Activity properties

possible calculations:

Absolute value

Needs an integer and returns the absolute value

Example:

IntegerValue = -10

```
begin
  var ResultValue: Int64;
  ResultValue := Abs(IntegerValue);
end;
```

Result = 10

Integer division

Needs a base value, a divisor, and value to put the remainder in, returns the result of the integer division

Example:

DivisorValue = 3

IntegerValue = 1000

```
begin
  var ResultValue: Int64;
  var WordRemainder: Word;
```

```
var WordResult: Word := 0;  
DivMod(IntegerValue, DivisorValue, WordResult, WordRemainder);  
ResultValue := WordResult;  
Remainder := WordRemainder;  
end;
```

ResultValue = 333

Remainder = 1

Average value

Needs an array of *Single, Extended or Double*. Returns the average of all values in an array.

Example:

SingleArray = [0,1,2,3,4,5]

```
begin  
  var ResultValue: Double;  
  ResultValue := Mean(SingleArray);  
end;
```

ResultValue = 2.5

Power

Raises Base to any Exponent power.

Example:

IntegerValue = 3

DecimalValue = 7.5

```
begin  
  var ResultValue: Double;  
  ResultValue := Power(DecimalValue, IntegerValue);  
end;
```

ResultValue = 421.875

Squared

Needs an integer and returns the squared value.

Example:

DecimalValue = 7.5

```
begin  
  var ResultValue: Double;  
  ResultValue := Sqr(DecimalValue);  
end;
```

ResultValue = 56.25

Square root

Needs an integer and returns the square root value.

Example:

DecimalValue = 56.25

```
begin
  var ResultValue: Double;
  ResultValue := Sqrt(DecimalValue);
end;
```

ResultValue = 7.5

[Log base 10](#)

Calculates log base 10 of the provided integer or double.

Example:

IntegerValue = 100

```
begin
  var ResultValue: Double;
  ResultValue := Log10(IntegerValue);
end;
```

ResultValue = 2

[Log base 2](#)

Calculates log base 2 of the provided integer or double.

Example:

IntegerValue = 16

```
begin
  var ResultValue: Double;
  ResultValue := Log2(IntegerValue);
end;
```

ResultValue = 4

[Log base N](#)

Calculates log base (provided base integer) of the provided integer or double.

Example:

IntegerValue = 27

BaseValue = 3

```
begin
  var ResultValue: Double;
  ResultValue := LogN(BaseValue, IntegerValue);
end;
```

ResultValue = 3

Natural logarithm

Returns the value of Log to the natural base (2.718281828459).

Example:

IntegerValue = 20

```
begin
  var ResultValue: Double;
  ResultValue := Ln(IntegerValue);
end;
```

ResultValue = 2.9995732...

Natural logarithm of (X+1)

Returns the natural log of (X+1)

Example:

IntegerValue = 19

```
begin
  var ResultValue: Double;
  ResultValue := Ln(IntegerValue);
end;
```

ResultValue = 2.9995732...

Natural logarithms raised to power

Returns the exponential of a provided number to the natural base (2.718281828459).

Example:

IntegerValue = 5

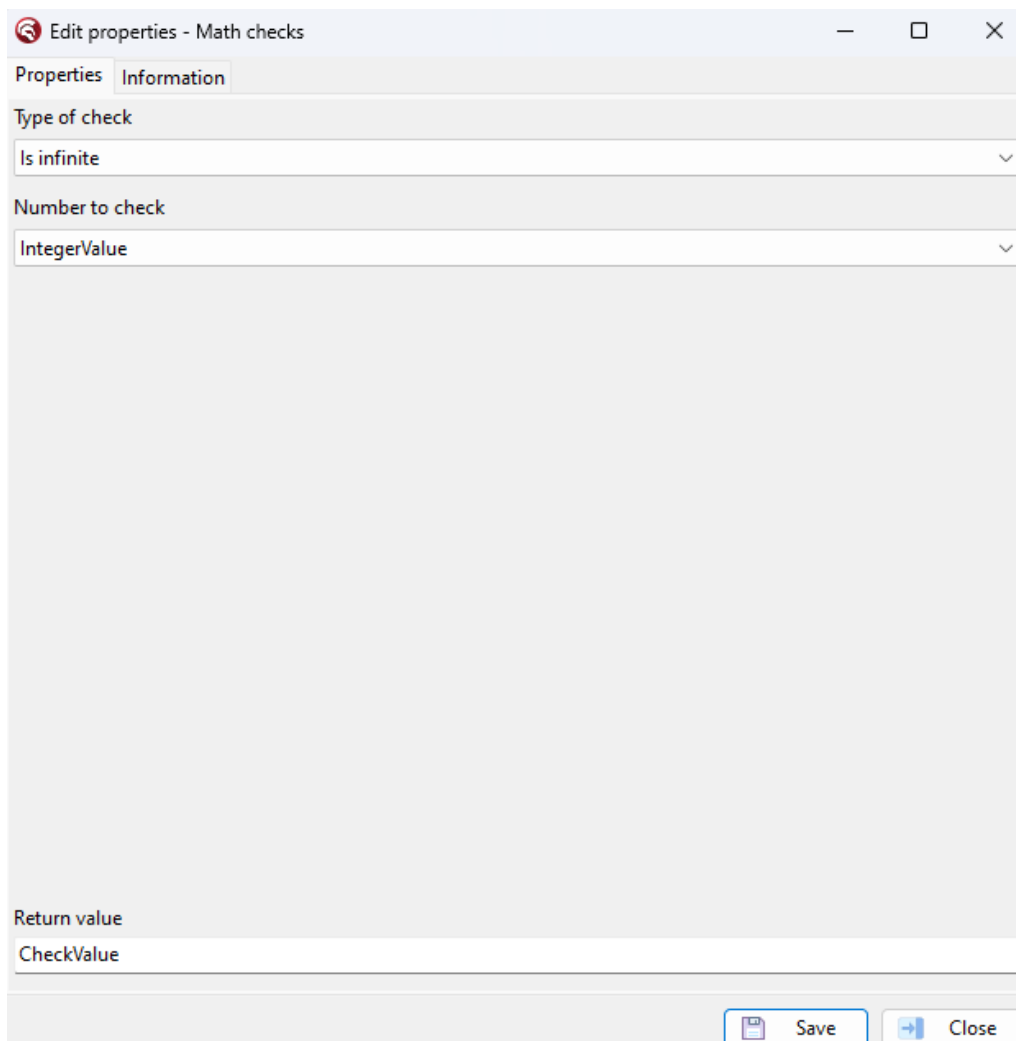
```
begin
  var ResultValue: Double;
  ResultValue := Exp(IntegerValue);
end;
```

ResultValue = 148,413

7.14.2 Checks

The Checks activity returns a boolean value for a validation check of a number.

uses [*System.Math*](#)



Edit properties - Math checks

Properties Information

Type of check

Is infinite

Number to check

IntegerValue

Return value

CheckValue

Save Close

Activity properties

Possible checks:

[Is Infinite](#)

Indicates when a variable represents an infinite value.

Example:

IntegerValue = 10

```
begin
  var CheckValue: Boolean;
  CheckValue := IsInfinite(IntegerValue);
end;
```

Result = False

[Is not a number](#)

Indicates when a variable represents a 'Not a number' (Nan) value.

Example:

IntegerValue = 10

```
begin
  var CheckValue: Boolean;
  CheckValue := IsNan(IntegerValue);
end;
```

Result = False

Is zero

Indicates when a floating-point variable or expression evaluates to zero with a deviation.

Example:

DecimalValue = 0.5

Deviation = 1

```
begin
  var CheckValue: Boolean;
  CheckValue := IsZero(DecimalValue, 1);
end;
```

Result = True

Is same value

Indicates whether two floating-point values are equal. with a possible deviation.

Example:

IntegerValue = 2

DecimalValue = 0.5

Deviation = 1

```
begin
  var CheckValue: Boolean;
  CheckValue := SameValue(DecimalValue, IntegerValue, 1);
end;
```

Result = False

Is positive value

Indicates whether a numeric value is positive.

Example:

IntegerValue = 10

```
begin
  var CheckValue: Boolean;
  var SignValue := Sign(IntegerValue);
  CheckValue := SignValue = TValueSign(PositiveValue);
```

```
end;
```

Result = True

Is negative value

Indicates whether a numeric value is negative.

Example:

IntegerValue = 10

```
begin
  var CheckValue: Boolean;
  var SignValue := Sign(IntegerValue);
  CheckValue := SignValue = TValueSign(NegativeValue);
end;
```

Result = False

In range

Indicates whether a value falls within a specified range.

Example:

IntegerValue = 2

MinRange = 1

MaxRange = 3

```
begin
  var CheckValue: Boolean;
  CheckValue := InRange(IntegerValue, 1, 3);
end;
```

Result = True

Providing a higher minimum than maximum is possible, but is also never true.

7.14.3 Finance

The finance activity includes the financial activities delphi has to offer.

uses [Business And Finance Routines](#)

Edit properties - Math finance

Properties Information

Finance type
Interest rate

Number of periods
3

Payment time
End of period

Present value
DecimalValue

Future value
DecimalValue1

Payment value
DecimalValue2

Return value
Finance

Save Close

Activity properties

Possible financial functions:

Double declining balance

Calculates the depreciation of an asset using the double-declining balance method.

Example:

Cost = 5000

Salvage = 1000

Life expectancy = 5

Period = 2

```
begin
  var Finance: Double;
  Finance := DoubleDecliningBalance(Cost, Salvage, 5, 2);
end;
```

Result = 2400

Future investment value

Calculates the future value of an investment.

Example:

Present value = 1000

Payment = 100

Rate value = 0.05

Period = 5

Payment time = End of period

```
begin
  var Finance: Double;
  Finance := FutureValue(RateValue, 5, Payment, PresentValue,
    ptEndOfPeriod);
end;
```

Result = -1828.84 (value is negative to symbolize what you could withdraw)

Interest rate

Returns the interest rate required to increase PresentValue to FutureValue.

Present value = 1000

Payment = 0

Future value = -10000 (*needs to be negative to symbolize what you could withdraw*)

Period = 10

Payment time = End of period

```
begin
  var Finance: Double;
  Finance := InterestRate(RateValue, 5, Payment, PresentValue,
    ptEndOfPeriod);
end;
```

Result = 0.259...

Internal rate of return

Calculate the IRR of a cashflow with a guess in case of positive and negative cashflows.

CashFlow = [-1000,250,250,250,250,250,250] (*First value is negative to symbolize the investment*)

GuessValue = 0.2

```
begin
  var Finance: Double;
  Finance := InternalRateOfReturn(GuessValue, CashFlow);
```

```
end;
```

Result = 0.12978...

Loan interest

Calculate the portion of a loan payment that reflects the interest.

Present value = 1000

Future value = 500

Period = 3

number of periods = 5

Rate = 0.05

Payment time = End of period

```
begin
  var Finance: Double;
  Finance := InterestPayment(Rate, 3, 5, PresentValue,
    FutureValue, ptEndOfPeriod);
end;
```

Result = -22.175...

Investment periods

Calculates the number of payment periods required for an investment of PresentValue to reach a value of FutureValue

Present value = 1000

Future value = -2000 (*needs to be negative to symbolize what you could withdraw*)

Payment = 50

Rate = 0.10

Payment time = End of period

```
begin
  var Finance: Double;
  Finance := NumberOfPeriods(Rate, Payment, PresentValue,
    FutureValue, ptStartOfPeriod);
end;
```

Result = 5.22335...

Fully amortized payment

Calculates the Payment needed for the amount of periods to get from present value to future value

Present value = 1000

Future value = -2000 (*needs to be negative to symbolize what you could withdraw*)

Periods = 5

Rate = 0.10

Payment time = End of period

```
begin
  var Finance: Double;
  Finance := NumberOfPeriods(Rate, 5, PresentValue, FutureValue,
    ptEndOfPeriod);
end;
```

Result = 63.79748...

Periodical payment

Calculates the principal amount from a full payment after a given period in a number of periods.

Present value = 0

Future value = -100 (*needs to be negative to symbolize what you could withdraw*)

Periods = 5

period = 3

Rate = 0.10

Payment time = End of period

```
begin
  var Finance: Double;
  Finance := PeriodPayment(Rate, 3, 5, PresentValue, FutureValue,
    ptEndOfPeriod);
end;
```

Result = 19.81949

Present investment value

Calculates the present value when the future value after an amount of periods is know.

Future value = -1000 (*needs to be negative to symbolize what you could withdraw*)

Periods = 3

Rate = 0.10

payment = 50

Payment time = ptStartOfPeriod

```
begin
  var Finance: Double;
  Finance := PresentValue(Rate, 3, PaymentValue, FutureValue,
    ptStartOfPeriod);
end;
```

Result = 614,53794

Net present investment value

Calculates the net present value for an investment, expected cashflows, and a rate

CashFlow = [-1000,400,400,400,400];

Rate = 0.20

```
begin
  var Finance: Double;
  Finance := NetPresentValue(Rate, CashFlow, ptStartOfPeriod);
end;
```

Result = 35.49...

7.14.4 Rounding

The rounding activities can be used to round numerical values into integers or specified decimals

Edit properties - Math rounding

Properties Information

Type of rounding

Round to decimals

Number to round

DecimalValue

Digits to round

3

Return value

RoundingValue

Save Close

Activity properties

Possible rounding options

Ceil

Rounds up to an integer.

Example:

DecimalValue = 3.3

```
begin
```

```
var RoundingValue: Integer;  
RoundingValue := Ceil(DecimalValue);  
end;
```

Result = 4

Floor

Rounds down to an integer.

Example:

DecimalValue = 3.3

```
begin  
var RoundingValue: Integer;  
RoundingValue := Floor(DecimalValue);  
end;
```

Result = 3

Fraction parts

Returns the part after the ',' from a decimal.

Example:

DecimalValue = 3.3

```
begin  
var RoundingValue: Integer;  
RoundingValue := Frac(DecimalValue);  
end;
```

Result = 0.3

round to tens

Rounds a floating-point value to a specified power of ten.

The 'Digits to round' will be a positive integer in the RoundTo function.

Example:

DecimalValue = 1234.1234

Digets to round = 2

```
begin  
var RoundingValue: Double;  
RoundingValue := RoundTo(DecimalValue, 2);  
end;
```

Result = 1200

Round to decimals

Rounds a floating-point value to a specified digit.

The 'Digits to round' will be a negative integer in the RoundTo function

Example:

DecimalValue = 1234.1234

Digits to round = 2

```
begin
  var RoundingValue: Double;
  RoundingValue := RoundTo(DecimalValue, 2);
end;
```

Result = 1234.12

Trunc

Drops everything behind the ',' part of a decimal without rounding.

Example:

DecimalValue = -3.56

```
begin
  var RoundingValue: Double;
  RoundingValue := Trunc(DecimalValue);
end;
```

Result = -3

Integer part

Returns the part before the ',' from a decimal.

Example:

DecimalValue = 3.6

```
begin
  var RoundingValue: Double;
  RoundingValue := Trunc(DecimalValue);
end;
```

Result = 3

7.15 Rest operation

The Rest operation activity can be used for a multitude of rest calls to send or retrieve data

Uses [System.Net.HttpClientComponent.TNetHttpClient](#)

Edit properties - REST operation

Properties Information

Operation: GET

Content-Type:

URL: 'http://worldtimeapi.org/api/timezone/Europe/Amsterdam'

Body Parameters Authentication Connection

Body:

Operation result type

☒ Content as string value ☐ HttpResponse entity

Return value: HttpResponse

Save Close

Activity properties

```
begin
  var Response: string;
  var Client := TNetHTTPClient.Create(nil);
  try
    var HttpResponse :=
Client.GET('http://worldtimeapi.org/api/timezone/Europe/Amsterdam',
nil, []);
    if (HttpResponse.StatusCode > 199) and (HttpResponse.StatusCode
< 300) then
      Response := HttpResponse.ContentAsString
    else
      raise ENetException.CreateFmt('%d %s%s',
[HttpResponse.StatusCode, HttpResponse.StatusText, sLineBreak,
HttpResponse.ContentAsString]);
    finally
      Client.Free;
    end;
  end;
```

Resulting code

The example shows a simple GET call to the WorldTimeAPI, but the activity offers a lot more options, let's go over them one by one.

[Click here](#) for an extensive explanation of this activity on our YouTube channel.

Operation

The HTTP Method that should be used in the call.

Possible options:

GET to retrieve information,

POST to send new information

DELETE to delete information stored elsewhere

PUT To update information stored elsewhere

PATCH to partially update information stored elsewhere.

Content-type

The type of content for the current call, in get calls this would be the content that the client can accept, in post calls this would be the type of content included in the request.

[The list of possible options](#)

URL

The URL to call, this can be a string, variable or expression, in the parameter section is explained how to use a string with parameters.

Body

The content of the current request this can be a string, variable or expression.

Parameters

A parameter can be added with a name and value.

There are 4 different types of parameters that can be used at the moment

1. Header

Adds a TNetHeader to the call.

2. Body (not allowed in GET Methods)

Adds a form data field to the request

3. Query

Adds a query paramter to the URL of the request

4. URL-Segment

Repaleces the par '{param name}' in the url with the value of the param.

e.g.

Body Parameters Authentication Connection		
Add Edit Delete		
Kind	Name	Value
► URL-SEGMENT	'area'	'Europa'

'<http://worldtimeapi.org/api/timezone/{area}/Amsterdam>' ->

```
var HttpResponse := Client.GET('http://worldtimeapi.org/api/timezone/'
+ 'Europa' + '/Amsterdam', nil, []);
```

Authentication

Option to include basic auth into the request with username and password.

Timeout

The maximum amount of time the rest call should wait for respond in milliseconds (Default 60000)

Connection

The secure protocols option of the request. Multiple options are possible.
Possible options: SSL v2, SSL v3, TLS v1.0, TLS v1.1, TLS v1.2, TLS v1.3

Pagination

Many API's that handle a lot of data use pagination. a part of the result given, with an indicator for more results.

To help you with retrieving data from these kind of API's Codolex offers the pagination tab. There are two ways to handle pagination, with record/rows numbers, and with page numbers.

Loop until end of pages/records can be turned on to retrieve all data in one activity. Leave this option off when retrieving only one page, or when retrieving multiple pages manually for partial retrieval.

The screenshot shows the 'Pagination' tab in a configuration interface. At the top, there are tabs for 'Body', 'Parameters', 'Authentication', 'Connection', 'Pagination', and 'Result'. The 'Pagination' tab is active. Below the tabs, the 'Pagination type' section has three radio buttons: 'None', 'Row/record numbers' (which is selected), and 'Pages'. Below this, there is a checked checkbox for 'Loop until end of pages/records'. The 'Request param for page size' section contains two text input fields and a 'Param type' dropdown menu. The 'Request param for row/record number' section also contains two text input fields and a 'Param type' dropdown menu. The 'Response param for next number' section has one text input field and a dropdown menu currently set to 'HEADER'. At the bottom, there is a section 'Return the response param value in' with a single text input field.

1. Records/rows

Request param for page size is the amount of records you want to receive from the API in one page. The first field is an editor for the name it has to have in the request, and the second for the value.

Request param for row/record number is the starting point from where the records should be returned, so if you already have the records somewhere, and you only want the new records, you can provide the amount of records you already have to receive new records. To receive all records, you should fill a 0 for the value. The first field is an editor for the name it has to have in the request, and the second for the value.

Response param for the next number is the name of the response header that contains the next records or total number. Provide this value when looping until the end of pages/records, so the code knows when to stop looping.

Return the response param in could be filled in with an integer variable. this variable is used to put the number or the next page or total records in. this could provide useful information when looping manually for partial retrieval.

2. Pages

The parameters for pages are more or less the same, but keep in mind that in this case its not the amount of records that is being worked with, but the amount of pages.

Logging

The logging tab can be used to set a log function that gets called when the request gets called.

To set the function, create a unit with the following class procedure

```
class procedure LogRestAction(const Client: TNetHTTPClient; const
URL: string; const Method: string; const Content: TStream; const
Headers: TNetHeaders);
```

units to use for this function are: *System.Classes, System.Net.URLClient, System.Net.HttpClientComponent;*

In this function you can use the parameters to write information to a log file. For example, at what time the url gets called.

Given the name of the unit **RestActivity.Logger** and the class **TRestLogger**, providing the function would look something like this:

Body	Parameters	Authentication	Connection	Pagination	Logging	Result
Log function						
TRestLogger.LogRestAction						
Log namespace						
RestActivity.Logger						

Response

Body	Parameters	Authentication	Connection	Pagination	Result
Operation result type					
<input checked="" type="radio"/> Content as string value <input type="radio"/> HttpResponseMessage entity <input type="radio"/> JSON value <input type="radio"/> Selected entity					

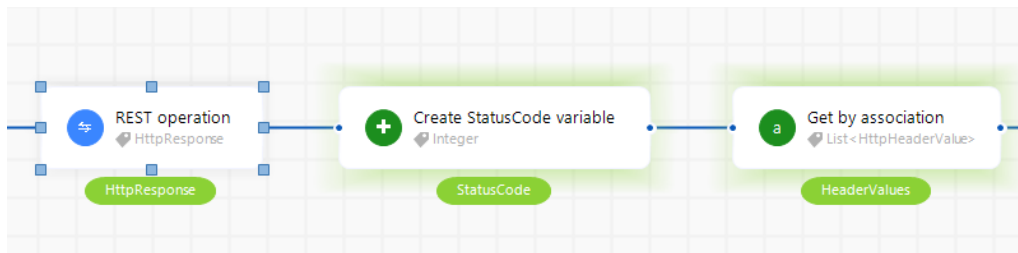
1. Content as string value.

The raw content of the HttpResponseMessage is parsed as string value in the result value

2. HttpResponseMessage entity

Its also possible to receive the result as an HttpResponseMessage Entity. This entity is added as [plugin datasource](#) entity by default in codolex.

This entity can be used to receive information about the rest result like 'Status code' and 'http headers'



3. JSON value

The content as string can also be directly parsed into a JSON Value. This value can then be used in the [JSON](#) ¹³² Activities to receive values.

This also has the option for a collection result, this option is mandatory when using pagination.

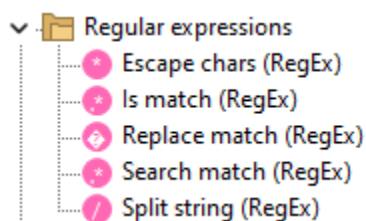
4. Selected entity

When you receive a JSON from the rest call as response, and the only thing that's needed from the JSON is a (list of an) entity, you can also directly parse it into the entity.

This also has the option for a collection result, this option is mandatory when using pagination.

7.16 Regular expressions

The reg-ex activities make it possible to do advanced search and replace actions on strings



More about the options available in reg-ex: [Options](#) ¹⁵⁶

[Escape chars](#) ¹⁵⁶

[IsMatch](#) ¹⁵⁷

[Split string](#) ¹⁵⁷

[Search match](#) ¹⁵⁸

[Replace match](#) ¹⁵⁹

Regex examples:

1. Email validation: `^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$`
2. Date in YYYY-MM-DD format: `^\d{4}-\d{2}-\d{2}$`

3. IP address validation: `^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$`
4. URL validation: `^(https?:\V)?([\da-z\.-]+)\.([a-z\.]{2,6})([\Vw \-]*)*V?$`
5. Extracting file extension: `\.[0-9a-z]+$`

7.16.1 Options

Most of the reg-ex activities have the following options possible

Options

<input type="checkbox"/> Compiled	<input type="checkbox"/> None
<input type="checkbox"/> Explicit capture	<input type="checkbox"/> Not empty
<input type="checkbox"/> Ignore case	<input type="checkbox"/> Single line
<input type="checkbox"/> Ignore pattern space	
<input type="checkbox"/> Multi line	

[System.RegularExpressions.TRegExOption](#)

7.16.2 Escape chars

Use the 'Escape chars' activity to replace special characters with their escape codes.

Uses [System.RegularExpressions.TRegEx.Escape](#)

Edit properties - Escape chars (RegEx)

Properties Information

String value

StringValue

Use wildcards

☒

Return value

EscapedStringValue

Save Close

Activity properties

```
begin
  var EscapedStringValue: string;
  EscapedStringValue := TRegEx.Escape(StringValue, True);
```

```
end;
```

Resulting Code

If 'Use wildcards' is checked, the '*' or '\?' characters are not converted.

7.16.3 IsMatch

The 'Ismatch' activity returns a boolean if a regex pattern returns a match on a string

Uses [System.RegularExpressions.TRegex.IsMatch](#)

Activity properties

```
begin
  var IsRegexMatch: Boolean;
  IsRegexMatch := TRegex.IsMatch(StringValue, RegexPattern, []);
end;
```

Resulting Code

7.16.4 Split string

Split string splits the string on regex matches and returns as a codolox string list.

Uses [System.RegularExpressions.TRegex.Split](#)

Activity properties

```
begin
  var SplitedRegex: ICodolexList<string>;
  var TextArray := TRegEx.Split(StringValue, RegexPattern, []);

  SplitedRegex := TCodolexList<string>.Create;
  SplitedRegex.AddRange(TextArray);
end;
```

Resulting Code

7.16.5 Search match

The 'search match' activity searches for a match in a string for a reg-ex pattern and returns the match.

Uses [System.RegularExpressions.TRegEx.Match](#)

Activity properties

```
begin
  var MatchResult: TMatch;
  MatchResult := TRegex.Match(StringValue, RegexPattern, []);
end;
```

Resulting Code

The activity returns an object of the **TMatch** variable. If the option 'Match collection' is chosen, the result will be a **TMatchCollection**

7.16.6 Replace match

The 'Replace match' replaces the matched part of a string to the replacement, and returns the new total as result.

Uses [System.RegularExpressions.TRegex.Replace](#)

Properties Information

RegEx pattern
RegexPattern

Match pattern
StringValue

Replacement
Replacement

Options

<input type="checkbox"/> Compiled	<input type="checkbox"/> None
<input type="checkbox"/> Explicit capture	<input type="checkbox"/> Not empty
<input type="checkbox"/> Ignore case	<input type="checkbox"/> Single line
<input type="checkbox"/> Ignore pattern space	
<input type="checkbox"/> Multi line	

Return value
Variable

Save Close

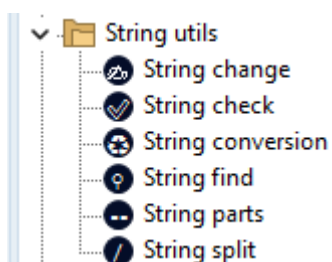
Activity properties

```
begin
  var Replacement: string;
  var Variable: string;
  Variable := TRegex.Replace(StringValue, RegexPattern,
    Replacement, []);
end;
```

Resulting Code

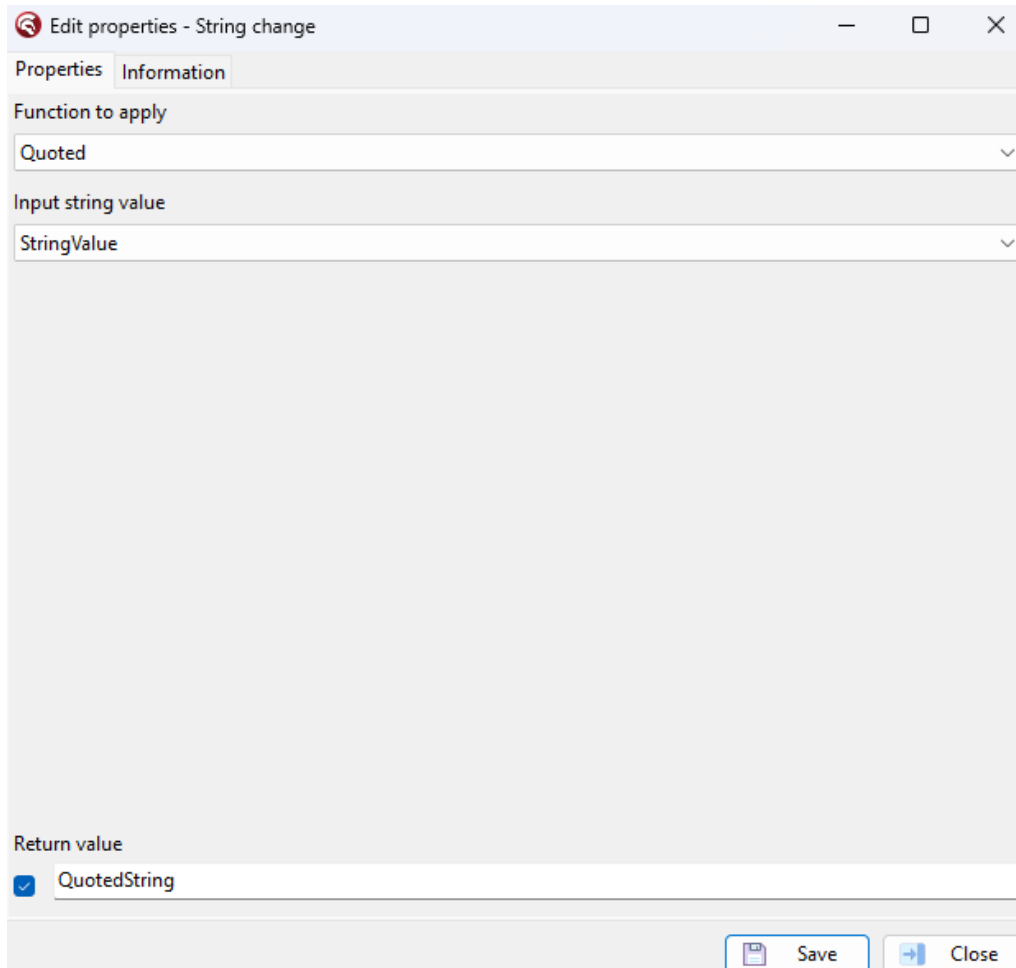
7.17 String utils

The string utils sections offers most of the stringutils functions available in delphi.



7.17.1 String change

The string change activity offers the delphi stringutils functions to change a string.



Activity properties

Return value is optional in some change functions.

If return value is selected, a new string with the changed value is created.

If return value is not selected, the old string is changed.

Possible change functions:

QuotedString

Adds quotes to the beginning and end of a string;

Example:

String: "The quick brown fox jumps over the lazy dog."

```
begin
  var ChangedString: string;
  ChangedString := StringValue.QuotedString;
end;
```

Result = "The quick brown fox jumps over the lazy dog."

DeQuotedString

Removes the quotes from a string at the beginning and the end if present.

Example:

String: "The quick brown fox jumps over the lazy dog."

```
begin
  var ChangedString: string;
  ChangedString := StringValue.DeQuotedString;
end;
```

Result = "The quick brown fox jumps over the lazy dog."

Insert

Inserts a string into a string at a given position.

Example:

String = "The quick brown fox jumps over the lazy dog."

String to insert = "test"

Position = 2

```
begin
  var ChangedString: string;
  ChangedString := StringValue.Insert(2, 'test');
end;
```

Result = "Thteste quick brown fox jumps over the lazy dog."

Join

Joins an array string with a separator.

Example:

Array = ['the', 'quick', 'brown'];

Separator = -

Position = 1

Number of items = 2

```
begin
  var ChangedString: string;
  var ArrayToJoin := StringArray;
  ChangedString := String.Join('-', ArrayToJoin, 1, 2);
end;
```

Result = "quick-brown"

Separator is optional

Position is optional with a default of 0

Number of items is optional with a default of everything after position.

Replace

Replaces a part of a string with another string.

Example:

String = "the-quick-brown"

Text to replace = '-'

new text = ' '

Replace all = True

Ignore case = False

```
begin
  var ChangedString: string;
  ChangedString := StringValue.Replace('-', ' ', [rfReplaceAll]);
end;
```

Result = *"the quick brown"*

Reverse

Reverses the complete string char by char.

Example:

String: "The quick brown fox jumps over the lazy dog."

```
begin
  var ChangedString: string;
  ChangedString := ReverseString(StringValue);
end;
```

Result = *"god yzal eht revo spmuj xof nworb kciuq ehT"*

LowerCase

converts all chars in a string to lowercase.

Example:

String: "The quick brOwn fox Jumps over The lazy dog."

```
begin
  var ChangedString: string;
  ChangedString := LowerCase(StringValue);
end;
```

Result = *"the quick brown fox jumps over the lazy dog."*

UpperCase

converts all chars in a string to uppercase.

Example:

String: "The quick brOwn fox Jumps over The lazy dog."

```
begin
  var ChangedString: string;
  ChangedString := LowerCase(StringValue);
end;
```

Result = "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG."

7.17.2 String check

The string check activity can be used to validate various things about strings

Edit properties - String check

Properties Information

String check function
Equals

Input string value
StringToCheck

String value to compare
StringToCompare

☒ Is case sensitive

Return value
IsSameString

Save Close

Activity properties

Most checks in the string check activity have to option to be case sensitive. This options results in different Delphi function, for example, **SameStr** if case sensitive, and **SameText** if case not sensitive when using the **Equals** function.

Possible checks:

[Equals / case insensitive](#)

Example:

StringToCheck = "Hello World"

StringToCompare = "world"

Is case sensitive = True

```
begin
  var var IsSameString: Boolean;
```

```
IsSameString := SameStr(StringToCompare, StringToCheck);  
end;
```

Result = False

[Contains](#) / [case insensitive](#)

Example:

StringToCheck = "Hello World"

StringToCompare = "world"

Is case sensitive = False

```
begin  
  var var IsSameString: Boolean;  
  IsSameString := ContainsText(StringToCompare, StringToCheck);  
end;
```

Result = True

[Ends with](#) / [case insensitive](#)

Example:

StringToCheck = "Hello World"

StringToCompare = "world"

Is case sensitive = True

```
begin  
  var var IsSameString: Boolean;  
  IsSameString := EndsStr(StringToCompare, StringToCheck);  
end;
```

Result = False

[Starts with](#) / [case insensitive](#)

Example:

StringToCheck = "Hello World"

StringToCompare = "hello"

Is case sensitive = False

```
begin  
  var var IsSameString: Boolean;  
  IsSameString := StartsStr(StringToCompare, StringToCheck);  
end;
```

Result = True

[Is empty](#)

Example:

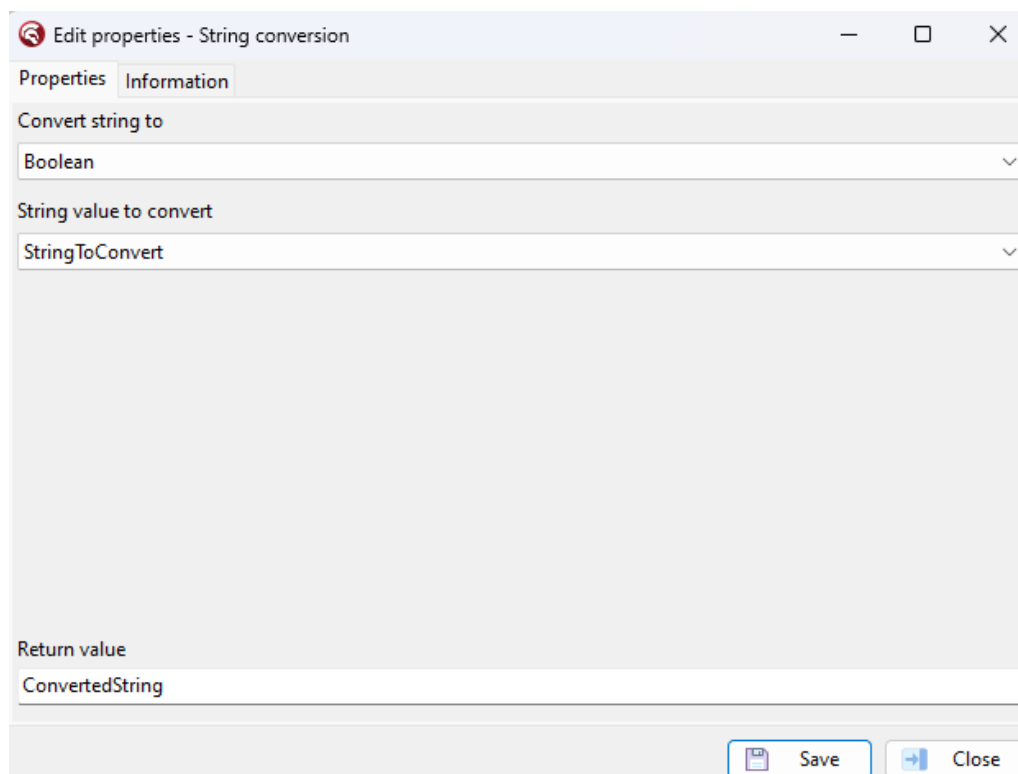
StringToCheck = "Hello World"

```
begin
  var var IsSameString: Boolean;
  IsSameString := StringToCheck.Trim.IsEmpty;
end;
```

Result = False

7.17.3 String conversion

The string conversion activity can be used to convert a string to another variable type or to upper/lower case the string.



Edit properties - String conversion

Properties Information

Convert string to

Boolean

String value to convert

StringToConvert

Return value

ConvertedString

Save Close

Activity properties

Possible variable types to convert to:

Boolean

Is true except when value is of the string is 'False'

Example:

String = "anything"

```
begin
  var ConvertedString: Boolean;
  ConvertedString := StringToConvert.ToBoolean;
end;
```

Result = True

CharArray (TArray<Char>)

Example:

String = "hello world"

```
begin
  var ConvertedString: TArray<Char>;
  ConvertedString := StringToConvert.ToCharArray;
end;
```

Result = ['H','e','l','l','o',' ','w','o','r','l','d']

Decimal (Double)

Example:

String = "3.5" (or "3,5" depending on settings)

```
begin
  var ConvertedString: Double;
  ConvertedString := StringToConvert.ToDouble;
end;
```

Result = 3.5

Extended

Example:

String = "3.5" (or "3,5" depending on settings)

```
begin
  var ConvertedString: Extended;
  ConvertedString := StringToConvert.ToExtended;
end;
```

Result = 3.5

Bigint (Int64)

Example:

String = "34214"

```
begin
  var ConvertedString: Int64;
  ConvertedString := StringToConvert.ToInt64;
end;
```

Result = 34214

Integer

Example:

String = "34214"

```
begin
  var ConvertedString: Integer;
  ConvertedString := StringToConvert.ToInteger;
end;
```

Result = 34214

Single

Example:

String = "34214"

```
begin
  var ConvertedString: Single;
  ConvertedString := StringToConvert.ToSingle;
end;
```

Result = 34214

Other conversions:

Lowercase

Example:

String = "Hello world"

```
begin
  var ConvertedString: Single;
  ConvertedString := StringToConvert.ToSingle;
end;
```

Result = hello world

Uppercase

Example:

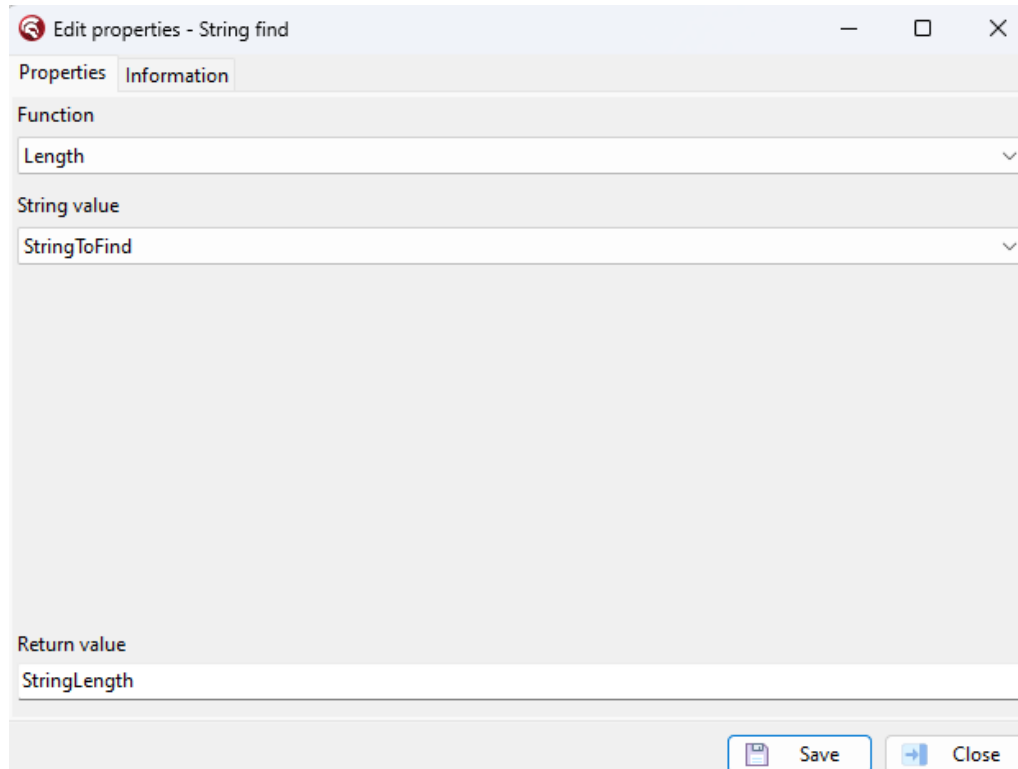
String = "Hello world"

```
begin
  var ConvertedString: Single;
  ConvertedString := StringToConvert.ToSingle;
end;
```

Result = HELLO WORLD

7.17.4 String find

The string find activity can be used to find the length or position of a sub string in a string



Edit properties - String find

Properties Information

Function

Length

String value

StringToFind

Return value

StringLength

Save Close

Activity properties

```
begin
  var StringLength: Integer;
  StringLength := StringToFind.Length;
end;
```

Resulting Code

String length

String length simply returns the length of the given string with the [.length](#) string helper

Finding the index of a substring

There are two options in finding a substring, the [first](#) or the last [index](#) of.

Both options have the following parameters:

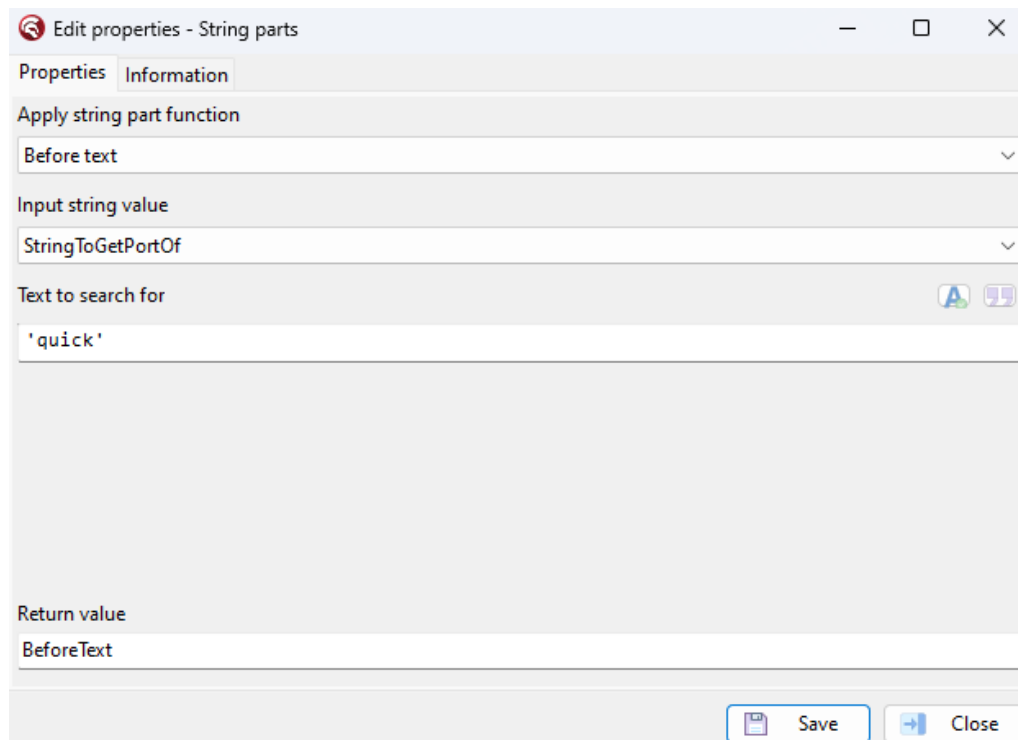
- String to find
- Initial offset
- The length of the substring.

The initial offset is the position of the start of the search. If left empty, the search will start at the beginning of the string.

The length of the substring determines how many chars needs to be searched as a maximum. If left empty, the search will go to the end of the string.

7.17.5 String parts

The string parts activity can be used to get different parts of the activity



Edit properties - String parts

Properties Information

Apply string part function

Before text

Input string value

StringToGetPortOf

Text to search for

'quick'

Return value

BeforeText

Save Close

Activity properties

Possible parts to search for in strings

Before text

All the text before a substring in a string, excluding the substring.

Example:

String: "The quick brown fox jumps over the lazy dog".

Search for: "quick".

```
begin
  var StringPartsResult: string;
  var Needle := 'quick';
  var TextPosition := StringValue.IndexOf(Needle);
  StringPartsResult := Copy(StringValue, 0, TextPosition);
end;
```

Result: "The ".

Beyond text

All the text after a sub-string in a string, excluding the sub-string.

Example:

String: "The quick brown fox jumps over the lazy dog".

Search for: "quick".

```
begin
  var StringPartsResult: string;
  var Needle := 'quick';
  var TextPosition := StringValue.IndexOf(Needle);

  StringPartsResult := EmptyStr;
  if TextPosition > -1 then
    begin
      var LengthToCopy := StringValue.Length - Needle.Length -
        TextPosition;
      TextPosition := TextPosition + Needle.Length + 1;
      StringPartsResult := Copy(StringValue, TextPosition,
        LengthToCopy);
    end;
  end;
```

Result: " brown fox jumps over the lazy dog ".

Duplicates

Duplicates the input string by given amount.

Example:

String: "brown fox"

Duplicate amount: 3

```
begin
  var StringPartsResult: string;
  StringPartsResult := DupeString(StringValue, 3);
end;
```

Result: "brown foxbrown foxbrown fox"

Left part

Returns a sub-string that contains a number of characters from the beginning of the string.

Example:

String: "The quick brown fox jumps over the lazy dog".

Left string amount: 7.

```
begin
  var StringPartsResult: string;
  StringPartsResult := LeftStr(StringValue, 7);
end;
```

Result: "The qui".

Mid part

Returns a sub-string that contains a number of characters from a starting position in the string.

Example:

String: "The quick brown fox jumps over the lazy dog".

Mid string amount: 7.

Starting position: 10.

```
begin
  var StringPartsResult: string;
  StringPartsResult := MidStr(StringValue, 10, 7);
end;
```

Result: " brown ".

Right part

Returns a sub-string that contains a number of characters from the end of the string.

Example:

String: "The quick brown fox jumps over the lazy dog".

Right string amount: 7.

```
begin
  var StringPartsResult: string;
  StringPartsResult := RightStr(StringValue, 7);
end;
```

Result: "azy dog".

Padding -> Right part & Left part

Adds a character to the end and/or front of a string a given amount of times.

Example:

String: "The quick brown fox jumps over the lazy dog".

Right string amount: 3.

Character to add: "-".

Add to: "Both"

```
begin
  var StringPartsResult: string;
  var PaddingLengthToAdd := StringValue.Length + 3;
  StringPartsResult := StringValue.PadLeft(PaddingLengthToAdd,
  '-');
  PaddingLengthToAdd := PaddingLengthToAdd + 3;
  StringPartsResult :=
  StringPartsResult.PadRight(PaddingLengthToAdd, '-');
end;
```

Result: "---The quick brown fox jumps over the lazy dog---".

Skip end

Removes a sub-string from a string based on a starting position and a number of characters to remove.

Example:

String: "The quick brown fox jumps over the lazy dog".

Starting position: 10.

Characters to remove: 10.

```
begin
  var StringPartsResult: string;
  StringPartsResult := StringValue.Remove(10, 10);
end;
```

Result: "The quick jumps over the lazy dog".

7.17.6 String split

The string split activity can be used to split strings on a delimiter and return the result as array

The screenshot shows a dialog box titled "Edit properties - String split". It has two tabs: "Properties" and "Information". Under the "Properties" tab, there are three sections: "String value" with a dropdown menu showing "StringToSplit", "Delimiter" with a text input field containing a comma, and "Return value" with a dropdown menu showing "SplitStringArray". At the bottom right, there are two buttons: "Save" and "Close".

Activity properties

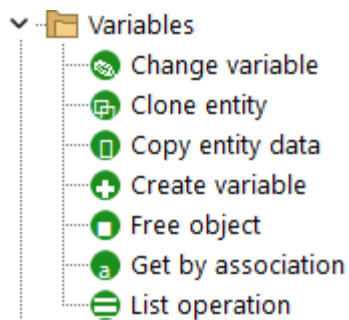
```
begin
  var SplitStringArray: ICodolexList<string>;
  SplitStringArray := TCodolexCollections.CreateList<String>;
```

```
var SplittedParts := SplitString(StringToSplit, ',');  
  
SplitStringArray.AddRange(SplittedParts);  
end;
```

Resulting Code

7.18 Variables

The variable activities can be used to create entities/variables and manipulate them.



[Create variable](#) ¹⁷⁴

[Change variable](#) ¹⁷⁸

[List Operations](#) ¹⁷⁸

[Copy entity data](#) ¹⁸⁰

[Free Object](#) ¹⁸¹

[Get by association](#) ¹⁸²

7.18.1 Create variable

The create variable activity can be used to create instances of ordinal types, lists, entities and classes. The result of this activity is a variable that can be used in the rest of the flow, and can be set as the result of a flow. It is possible to initialise the variable directly, by specifying text in the initialisation field.

Edit properties - Create variable

Properties | Information

Scope

☒ Declare local variable ☐ Initialise class variable

Data type

Array

Initialization

ExisitingStringArray

Collection type

String

Return value

StringArray

Save Close

Activity properties

```
begin
  var StringArray: TArray<string>;
  StringArray := ExisitingStringArray;
end;
```

Resulting Code

Possible variable types:

- Array -> TArray<T>
- Bigint -> Int64
- Binary -> ICodolexBinary
- Boolean -> Boolean
- Custom
- Date/Time -> TDateTime
- Decimal -> Double
- Entity
- Enumeration -> unsupported
- Flow class
- Integer -> Integer
- JSONValue -> TJSONValue
- List -> ICodolexList<T>
- String -> string

Arrays & Lists

Arrays and list have the extra option for collection type, this determines the T of the array/list.

Possible options are:

- Auto increment -> Integer

- Bigint -> Int64
- Boolean -> Boolean
- Custom
- Date/Time -> TDateTime
- Decimal -> Double
- Entity
- Integer -> Integer
- String -> string

Custom

A custom type variable can be any type needed, in the create variable activity, the custom type also give the option to include a unit in the implementation if needed for the variable.

The screenshot shows a configuration window for a 'Custom' data type. It has three main sections: 'Data type' with a dropdown menu set to 'Custom'; 'Custom type' with an empty text input field; and 'Namespace' with another empty text input field.

Entity

The entity type lets you create an entity and fill in the attributes. The possible entities are the entities available in the projects datasources.

The screenshot shows a configuration window for an 'Entity' data type. It includes a 'Data type' dropdown set to 'Entity', an 'Entity' dropdown set to 'Codolex.Shippers', and a 'Properties' section containing a table with three columns: Name, Type, and Value.

Name	Type	Value
CompanyName	String	'CreatedCompany'
Phone	String	
Orders_ShipVia_ShipperIDList	List<Orders>	OrdersList

Initialize class variable

When using class variables in a flow class, the initialisation is possible in every **non-class** flow of that class.

Edit properties - Create variable

Properties | Information

Scope

☐ Declare local variable ☒ Initialise class variable

Variable

- Class: FlowClassString

Initialization

'Initialised String'

Save Close

7.18.2 Clone entity

The clone entity activity creates a new instance of an existing entity.

Edit properties - Clone entity

Properties | Information

Entity to clone

Event

Return value

SecondEvent

Save Close

Activity properties

```
begin
  var SecondEvent: CodolexProject.DataSource.TestDB.IEvent;
  SecondEvent := CodolexProject.DataSource.TestDB.TEvent.Create;
  SecondEvent.EventId := Event.EventId;
  SecondEvent.CourseId := Event.CourseId;
  SecondEvent.LocationId := Event.LocationId;
  SecondEvent.EventDate := Event.EventDate;
  SecondEvent.poster := Event.poster;
  SecondEvent.Location := Event.Location;
  SecondEvent.Course := Event.Course;
  SecondEvent.EventRegistrations := Event.EventRegistrations;
end;
```

Resulting Code

Keep in mind that some properties that must be unique to entities in the database needs to be replaced with a different value.

Associations are included as well, although the associated objects are not cloned.

7.18.3 Change variable

The "change variable" activity is used to modify the value of a variable. To do this, an expression can be specified. An expression can be a simple text, but also a complex formula or a reference to another variable.

Edit properties - Change variable

Properties Information

Variable to change

Shippers

Properties

Name	Type	Value
CompanyName	String	'NameChange'
Phone	String	
Orders_ShipVia_Shipper	List<Orders>	OrdersList

Save Close

Activity properties

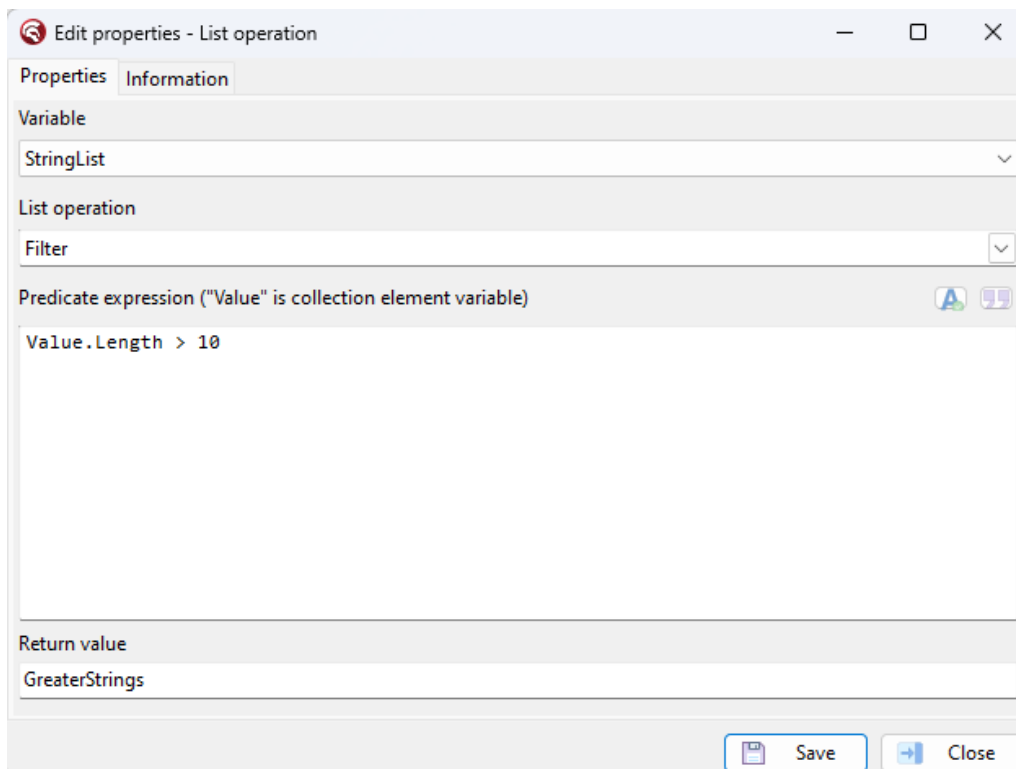
```
begin
  Shippers.CompanyName := 'NameChange';
  Shippers.Orders_ShipVia_ShipperIDList := OrdersList;
end;
```

Resulting Code

7.18.4 List Operations

Use the list operation activity to control list-related tasks.

Uses [System.Generics.Collections.TList](#)



Activity properties

```
begin
  var GreaterStrings: ICodolexList<string>;
  GreaterStrings := StringList.Filter(
    function(Value: string): Boolean
    begin
      Result := Value.Length > 10;
    end
  );
end;
```

Resulting Code

The list operation supports the following actions:

- Count
- Filter
- First
- Last
- Max
- Min
- Sum

Count

Counts the amount of items in a list.

Filter

Filter also needs an Predicate expression to filter on. Value is the collection element variable to be used in the expression.

First

Returns the first object in the collection.

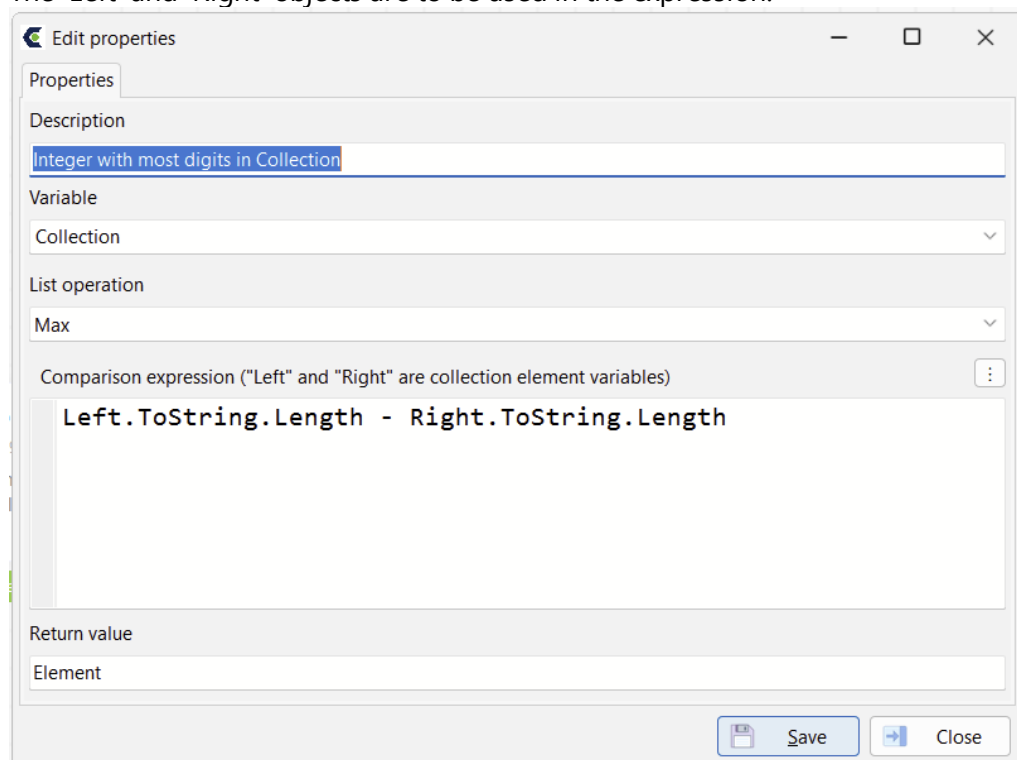
Last

Returns the last object in the collections.

Max

Returns the object with the max value of a given expression.

The 'Left' and 'Right' objects are to be used in the expression.



The screenshot shows a software interface titled "Edit properties". It contains several fields and a text area:

- Properties** (tab)
- Description**: A text field containing "Integer with most digits in Collection".
- Variable**: A dropdown menu showing "Collection".
- List operation**: A dropdown menu showing "Max".
- Comparison expression ("Left" and "Right" are collection element variables)**: A text area containing the expression `Left.ToString.Length - Right.ToString.Length`.
- Return value**: A dropdown menu showing "Element".
- At the bottom right, there are "Save" and "Close" buttons.

Min

Returns the object with the min value of a given expression.

The 'Left' and 'Right' objects are to be used in the expression.

Sum

Can only be used on a list with number types, returns the sum of all numbers in the list.

Sort

Changes the list to a certain order. It possible to select the attribute of an entity, or use a left and right comparison.

7.18.5 Copy entity data

The copy entity data activity can be used to transfer the values of an instance of an entity to another variable.

Edit properties - Copy entity data

Properties Information

Copy values from

FromEntity

To variable

ToEntity

Exclude fields

CompanyName

☒ Copy primary fields

Save Close

Activity properties

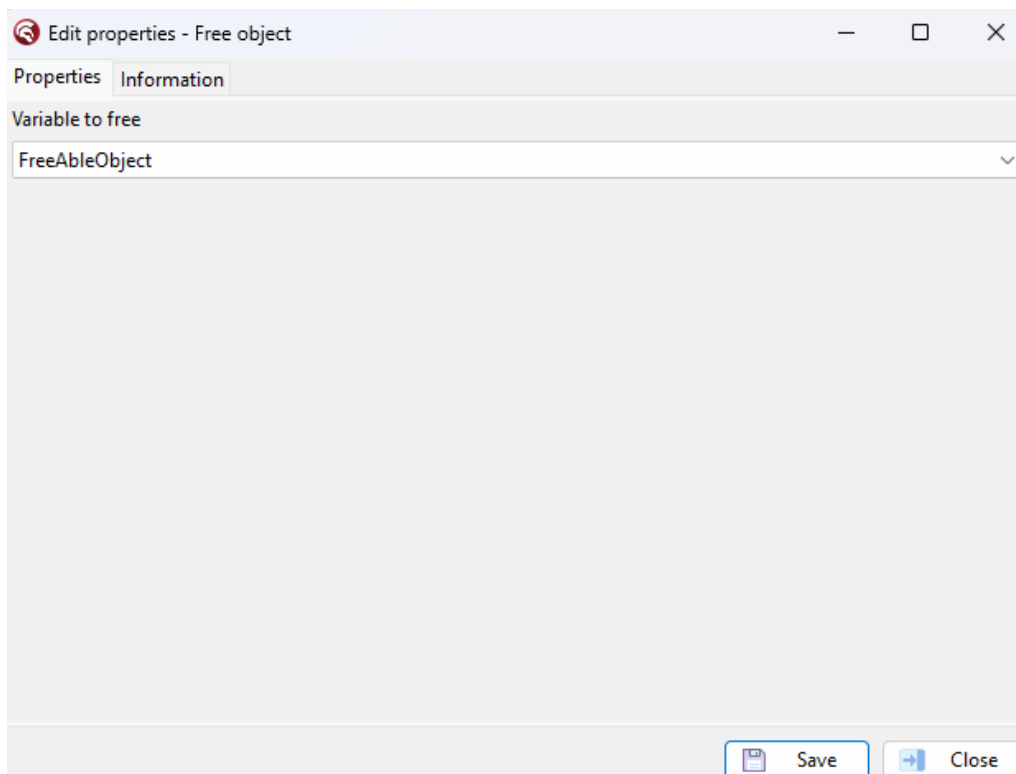
```
begin
  ToEntity.ShipperID := FromEntity.ShipperID.Value;
  ToEntity.Phone := FromEntity.Phone.Value;
end;
```

Resulting Code

You can exclude fields to be copied, separated by a comma.
Associations are not included in the copy activity.

7.18.6 Free Object

The Free object can be used to release the contents of a variable.
Uses [System.SysUtils.FreeAndNil](#)



Activity properties

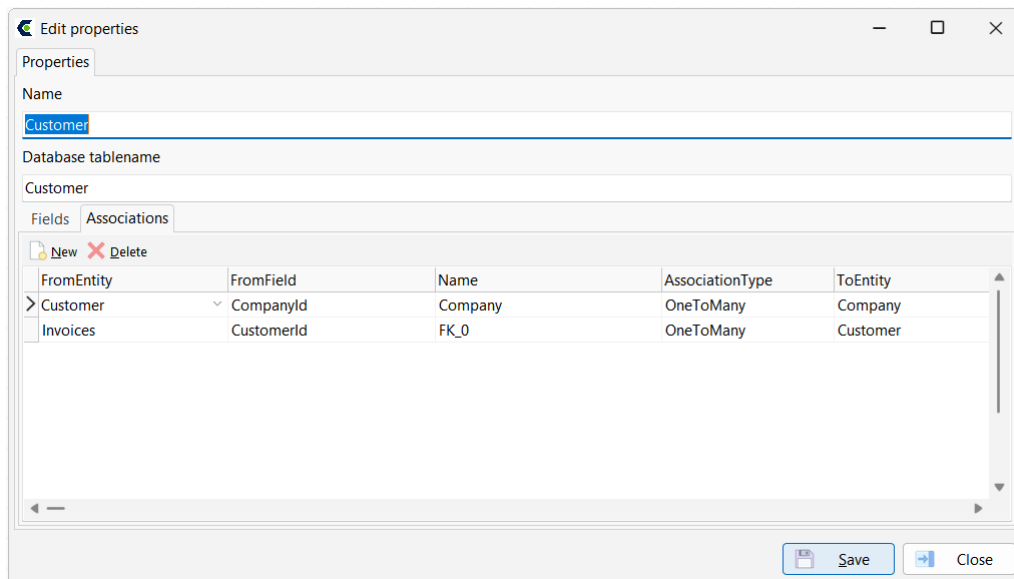
```
begin
    FreeAndNil(FreeAbleObject);
end;
```

Resulting Code

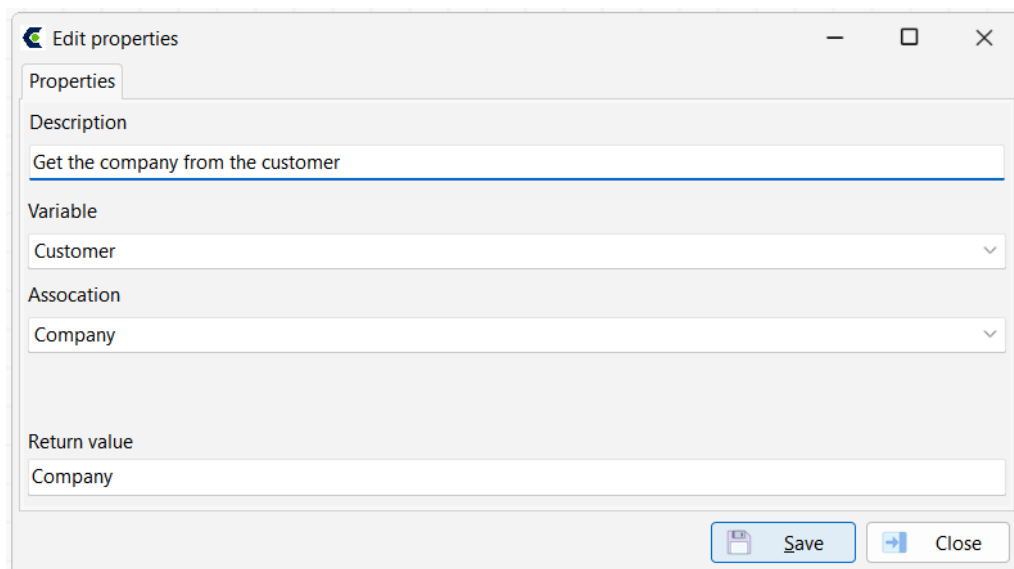
By default the entities created that are from a datasource are not available in the activity, The entities from datasources are interfaced objects by default and not available in the free object activity.
Custom entity types are available to be freed.

7.18.7 Get by association

The Get by association activity is used to create a new variable over an association of an entity with the value of the association.



The customer entity has an association with a company, as specified in the association tab of the entity. If you have an entity of the type Customer in your flow, you can get the corresponding Company by using the Get by association activity:



Activity properties

```
begin
  var Company: DataSource.Codolex.ICompany;
  Company := Customer.Company;
end;
```

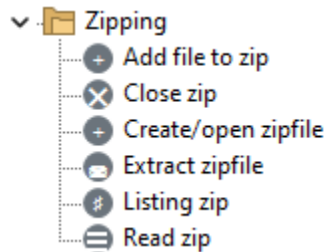
Resulting Code

Watch the following video for more information and a demo of this activity.

[Get by association](#)

7.19 Zipping

The zipping activities allows for easy creation and extraction of zip files.



[Create/open zip file](#) ¹⁸⁴

[Extract zipfile](#) ¹⁸⁵

[Add to zipfile](#) ¹⁸⁶

[Close zipfile](#) ¹⁸⁸

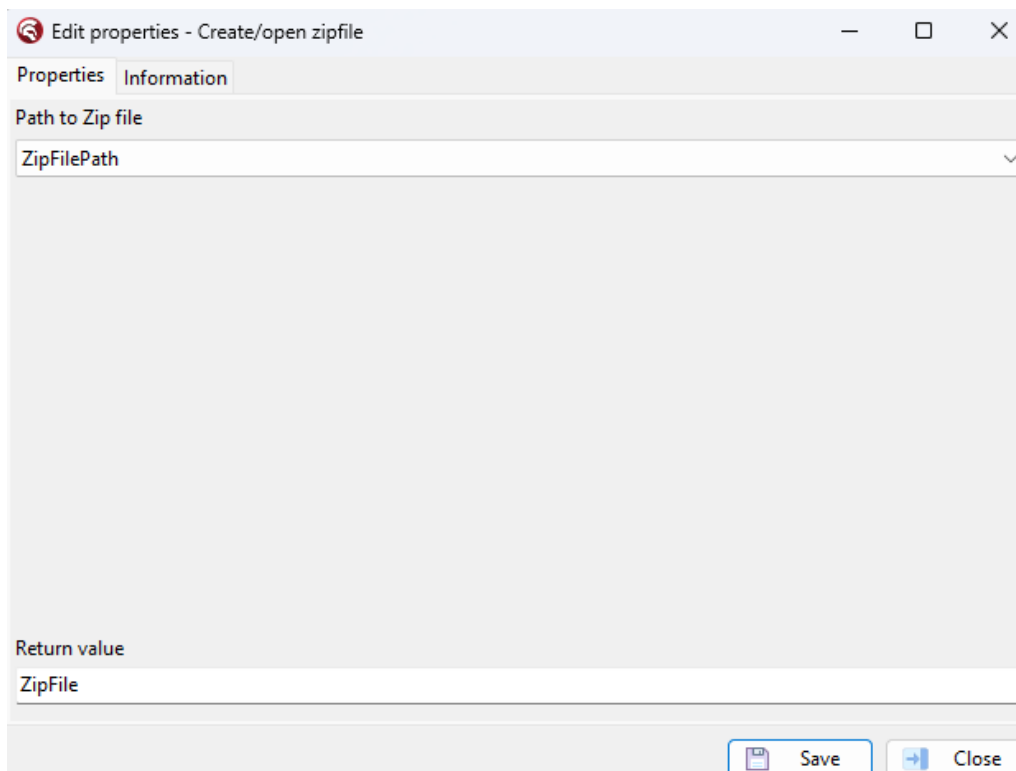
[Listing zip](#) ¹⁸⁸

[Read zip](#) ¹⁸⁹

7.19.1 Create/open zip file

The 'Create/open zip file' activity can be used to create a TZipfile object with a reference to an existing .zip file or create a new .zip file.

Uses [System.Zip.TZipFile](#)



Activity properties

begin

```
var ZipFile: TZipFile;  
var ZipPath: string;  
ZipPath := ZipFilePath;  
  
ZipFile := TZipFile.Create;  
ZipFile.Open(ZipPath, zmWrite);  
end;
```

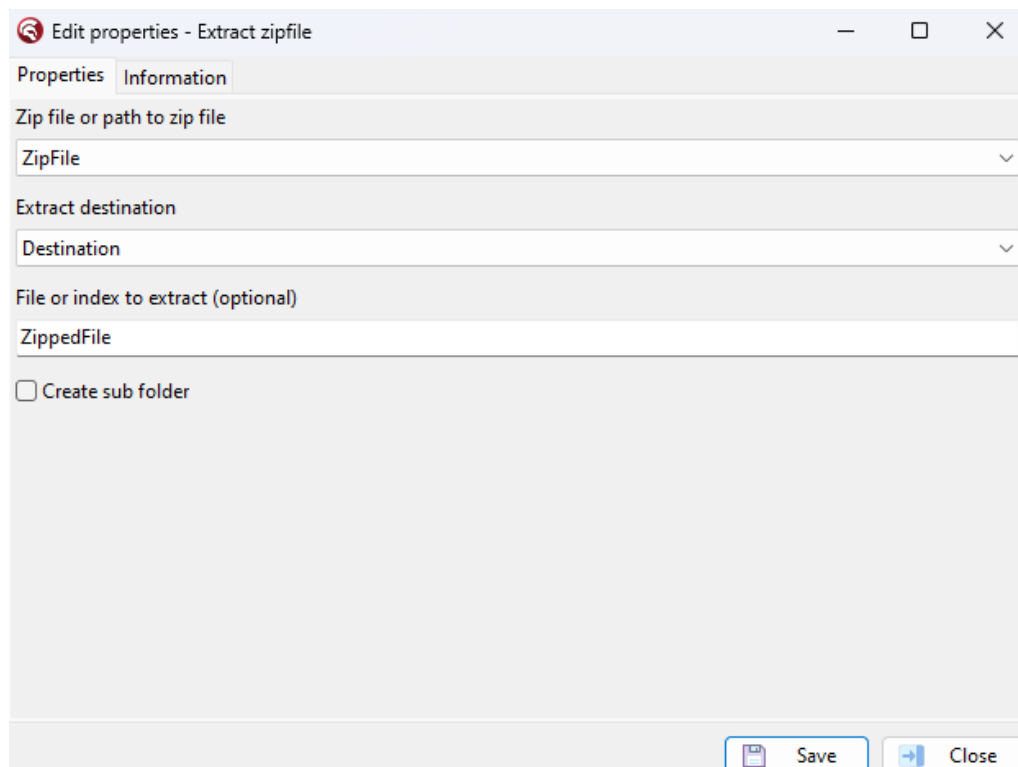
Resulting Code

If no .zip file is found at the zip path, and the path is valid, a new .zip file will be created.

7.19.2 Extract zipfile

Extract zipfile can be used to extract files from a zip, and place it at a destination

Uses [System.Zip.TZipFile.Extract](#)



Activity properties

```
begin  
  var CurrentZip := ZipFile;  
  CurrentZip.Extract('ZippedFile', Destination, False);  
end;
```

Resulting Code

The provided zipfile can be a string with the path, or a variable of the TZipFile type.

If a existing TZipFile variable is provided, the File must be Opened before extracting.

The destination must be an existing path to a folder.

If a file (string) or index (Integer) is provided, only the file with the name or location will be extracted.

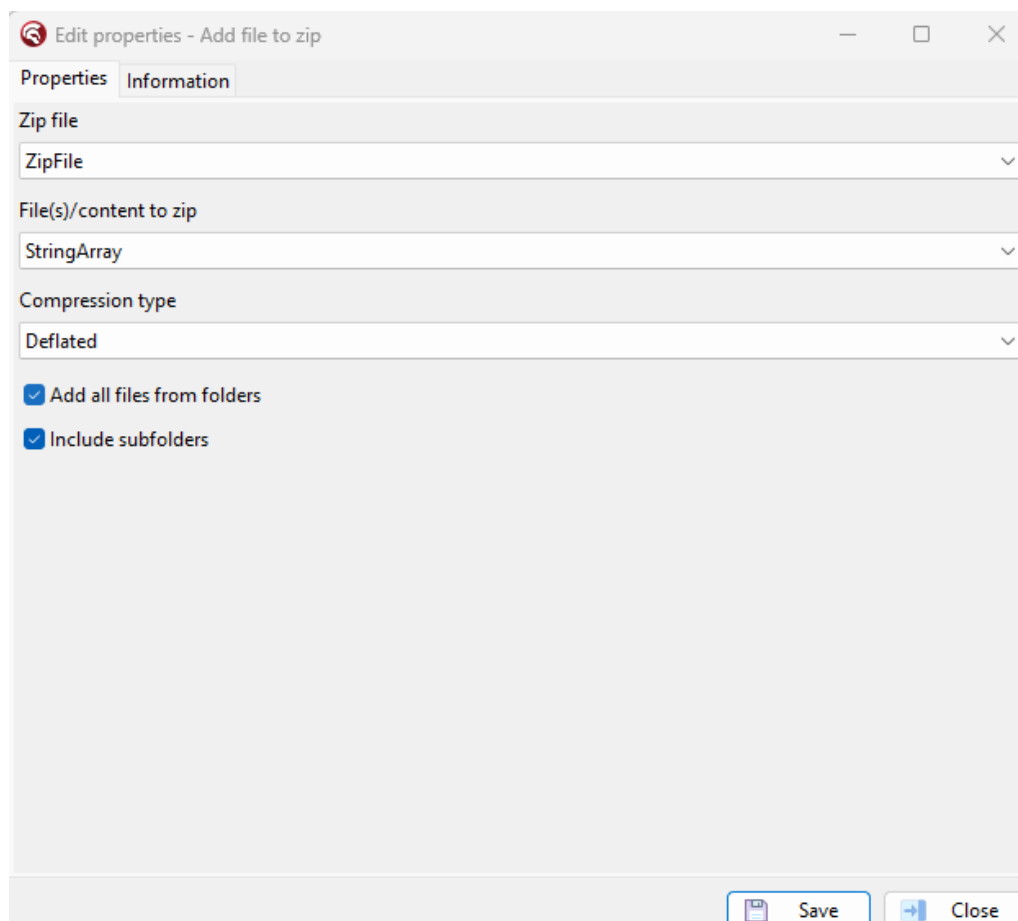
When 'Create sub folder' is selected, the path must be included in the Destination variable.

To prevent errors in the zip file, always [Close the zipfile](#)¹⁸⁸ after extracting from the zip.

7.19.3 Add to zipfile

Add to zipfile can be used to add files or directories to an open zip file.

Uses [System.Zip.TZipFile.Add](#)



Activity properties

```
begin
  var ArchiveName := '';
  var ZipFile1 := ZipFile;
```

```

for var ContentPath in StringArray do
begin
    var IsDirectory := TDirectory.Exists(ContentPath);
    if IsDirectory then
    begin
        var FolderInArchive := TPath.GetFileName(ContentPath) + '/';
        ZipFile1.Add(TBytes(nil), FolderInArchive, zcDeflate);

        var Files := TStringList.Create;
        Files.AddStrings(TDirectory.GetFiles(ContentPath));

        var FileItem: string;
        for FileItem in Files do
        begin
            ArchiveName := TPath.Combine(FolderInArchive,
            TPath.GetFileName(FileItem));
            ZipFile1.Add(FileItem, ArchiveName, zcDeflate);
        end;

        var SubFolders := TStringList.Create;
        SubFolders.AddStrings(TDirectory.GetDirectories(ContentPath,
        ' *.*', TSearchOption.soAllDirectories));

        for var SubFolderItem in SubFolders do
        begin
            Files.Clear;
            Files.AddStrings(TDirectory.GetFiles(SubFolderItem));

            var SubFolderName := TPath.GetFileName(SubFolderItem) +
            '/';
            var SubFolderArchiveName := TPath.Combine(FolderInArchive,
            SubFolderName);
            ZipFile1.Add(TBytes(nil), SubFolderArchiveName, zcDeflate);

            for FileItem in Files do
            begin
                ArchiveName := TPath.Combine(SubFolderArchiveName,
                TPath.GetFileName(FileItem));
                ZipFile1.Add(FileItem, ArchiveName, zcDeflate);
            end;
        end;
    end
else
begin
    ArchiveName := TPath.GetFileName(ContentPath);
    ZipFile1.Add(ContentPath, ArchiveName, zcDeflate);
end;
end;
end;

```

Resulting Code

The provided zipfile can be a string with the path, or a variable of the TZipFile type.

If a existing TZipFile variable is provided, the File must be Opened before adding.

'Files/content to add' can be a string or an array of strings, the paths can be folders or files.

The compression method is Deflate by default.

More about compression methods can be found in the embarcadero wiki:

[System.Zip.TZipCompression](#)

When adding folders, there are 2 options to consider.

If only the option to 'add all files from folder' is checked, only the files directly in the folder are added to the zip.

To include folders and files in those folders, the option 'Include subfolders' are also checked.

To prevent errors in the zip file, always [Close the zipfile](#) after adding to the zip.

7.19.4 Close zipfile

The close ZipFile must be used after extracting or adding data to a .zip file

Uses [System.Zip.TZipFile.Close](#)



Activity properties

```
begin
  ZipFile.Close;
end;
```

Resulting Code

7.19.5 Listing zip

'Listing zip' can be used to list all the files that are present in the zip file.

Uses [System.Zip.TZipFile.Read](#)

Activity properties

```
begin
  var ListedZip: ICodolexList<string>;
  var ZipFile := TZipFile.Create;
  try
    ZipFile.Open(ZipFileToUse, zmRead);
    ListedZip := TCodolexList<string>.Create;
    ListedZip.AddRange(ZipFile.FileNames);
  finally
    ZipFile.Close;
    ZipFile.Free;
  end;
end;
```

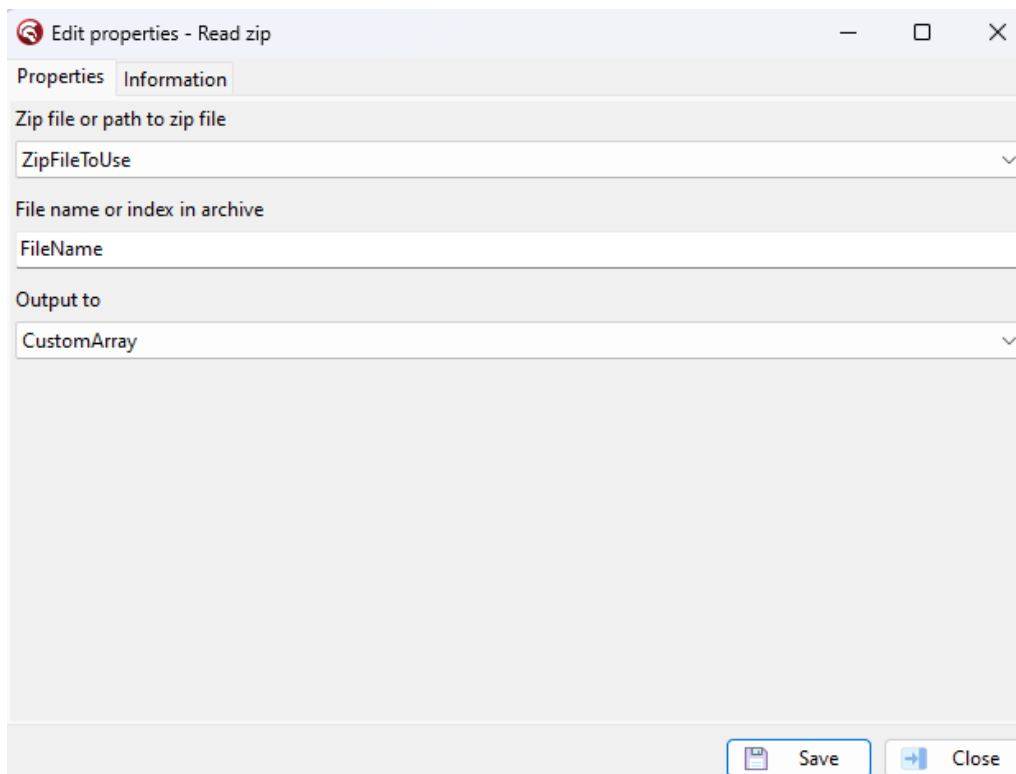
Resulting Code

The provided zipfile can be a string with the path, or a variable of the TZipFile type.

The result will be a TCodolexList of strings with the relative paths.
If a file (string) or index (Integer) is provided, only the file with the name or location will be listed as string.

7.19.6 Read zip

The read zip activity can be used to read the contents of a file in a zipfile without extracting.



Activity properties

```
begin
  ZipFileToUse.Read('FileName', CustomArray);
end;
```

Resulting Code

The output must be a variable of type *TBytes*, *TStream* or *TArray<Byte>*.

7.20 Advanced

Enter topic text here.

7.20.1 OAuth2

Requirements

- Paid access to the oauth2 component

OAuthComponent

Edit properties - Get authorization code

Properties Information

Select predefined provider

Microsoft

Authentication endpoint

'https://login.microsoftonline.com/' + TenantID + '/oauth2/v2.0/authorize'

Response type

☒ Code ☐ Token

Redirect is handled by

☐ Existing server ☒ Client side (embedded localhost is used)

HTTP type

☒ HTTP ☐ HTTPS

Redirect URL

http://localhost:7070

Port number (Uses a random port number if it's empty)

7070

Timeout when waiting for response (in milliseconds)

120000

Client ID

ClientID

Scope

'Directory.ReadWrite.All'

State

State

Return value

AuthorizationCode

Save Close

The component has quite a few options, let's go over them one by one

- Description

Used to set a description in the activity

- Predefined provider

There are some predefined providers added to the component, for example Microsoft. Selecting one of the providers fills the authentication endpoint with the usual endpoint.

E.g. for microsoft this is '<https://login.microsoftonline.com/{tenant}/oauth2/v2.0/authorize>' the only thing you need to replace is the tenant id in this case. Keep in mind that Codolox generates code, so any hardcoded sensitive information added here, is also added to the resulting code.

- Authentication endpoint

this is the endpoint that is going to be called when authentication is started. Check with your provider what the URL needs to be. Or select one of the predefined providers, and add the missing information.

- Response type

The option if the authorization endpoint should return a code, or a token. This defines how the result should be handled, so let's go over both options quickly.

Code:

The code is the first step in authentication. The Endpoint will send a code to the redirect url. This code must be sent to a second URL with the REST Activity to retrieve a token.

Token:

This retrieves a token directly, this token can be used in the calls to retrieve or send information.

- Redirect is handled by:

This defines how the code should be received.

If **Existing server** is selected, Codolex assumes the result will be sent to the given **Redirect url**, and no further handling of the authentication should be done in this activity. There is no result value in this case.

If **Client side** is selected, Codolex will start a server on the localhost which waits for the code/token result before going on with the rest of the activities in the flow. The result value will be the code if the authentication was successful.

- Redirect URL

The redirect url is the url where the code/token will be sent to. If **Existing sever** is selected for handling, the url can be filled in and should be available and able to handle the result code. If **Client side** is selected the redirect is always the localhost and cannot be changed, except for the port.

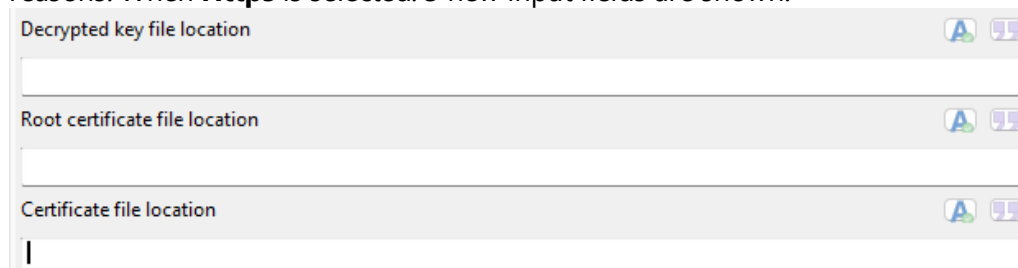
The result code is usually sent as a parameter, so the call to expect can look something like this:

['https://localhost:7070?code=12345abcd'](https://localhost:7070?code=12345abcd)

This URL must be configured at the provider

- Http type

Only available when **Client side** is selected. This provides the option to use a secured local server. Some providers only redirect to a Https url for security reasons. When **Https** is selected. 3 new input fields are shown.



The screenshot shows a configuration window with three input fields, each with a label and a search icon (magnifying glass) to its right:

- Decrypted key file location
- Root certificate file location
- Certificate file location

The first two fields are empty. The third field, 'Certificate file location', has a single character 'I' entered.

This allows to provide a certificate and a key to the activity to start a local server for the redirect URL.

See <https://www.openssl.org/> for more information about ssl.

- Port

Only available when **Client side** is selected. This defines the port of the redirect url where the provider can send the code to.

- Timeout

Only available when **Client side** is selected. This defines how long the activity should wait (in milliseconds) for the response. Don't make this number too low, given that the user needs some time to log-in in most cases.

- Scope

Some providers need a scope when requesting a code, the scope determines what rights the user has with the resulting code/token. This scope also needs to be given in the configuration for the provider in most cases.

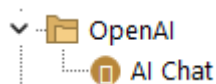
- State

The state can be used to pass extra information to the request, which then gets passed to the redirect url. This can be used when multiple calls come through at the redirect url, and some extra information is needed to find the caller for example. Another example is pagination.

The state is not required.

7.21 OpenAI

Open AI chat contains the activity to connect with ChatGPT



[AI Chat](#) ¹⁹³

7.21.1 AI Chat

The AI Chat activity lets you connect to the open AI api to make use of Chat GTP in your codolex project.

Uses [System.Net.HttpClientComponent.TNetHttpClient](#)

Edit properties - AI Chat

Properties Information

Model
GPT 4o

Authorization key
'YourApiKey'

List of all chat messages
ChatMessageList

Role
User

Chat message text
'Hello AI world'

Return value
ChatMessage

Save Close

Activity properties

```
begin
...
  try
    NetClient := THTTPClient.Create;
    NetClient.CustHeaders.Add('Authorization', 'Bearer ' +
'YourApiKey');
    NetClient.ContentType := 'application/json';

    var Response :=
NetClient.Post('https://api.openai.com/v1/chat/completions',
JsonStream);

    var IsSuccessful := (Response.StatusCode > 199) and
(Response.StatusCode < 300);
    if not IsSuccessful then
      raise ENetException.CreateFmt('%d %s%s',
[Response.StatusCode, Response.StatusText, sLineBreak,
Response.ContentAsString]);
```

```
    JsonResponse :=  
    TJSONValue.ParseJSONValue(Response.ContentAsString);  
  
    var ChoiceJson :=  
    JsonResponse.GetValue<TJSONArray>('choices');  
    var MessageJson :=  
    ChoiceJson[0].GetValue<TJsonObject>('message');  
  
    ChatMessage := JsonAdapter.CreateEntity;  
    JsonAdapter.MapToEntity(MessageJson, ChatMessage);  
    ...  
end;
```

Part of resulting Code

More information about the API:

<https://openai.com/api/>

The model can be set to one of the 5 options, Choosing the right model for your needs and token usage is recommended.

- gpt-4o (default)
- gpt-4o-mini
- gpt-4-turbo
- gpt-4
- gpt-3.5-turbo

The role can also be selected, depending on the type of conversation needed.

more about roles: <https://platform.openai.com/docs/guides/chat-completions/overview>

Provide a list of messages if you want to keep a conversation instead of just one question. this needs to be a list of the plugin datasource entity 'ChatMessage'. Keeping track of the conversation is important for asking follow-up questions, so the AI knows what you are talking about.

The question and response messages are directly added in the activity.

'Chat message' and 'Authorization key' are string fields that can be filled with the expression editor.

Creating your own activity

8 Creating your own activity

Here we will explain how to create your own custom Codolex activity. We will use an example based on two new activities: "Play sound" and "Stop sound".

[Getting started](#) ¹⁹⁷

[Implementation](#) ¹⁹⁹

[Tags](#) ²⁰⁷

[Defined entity](#) ²⁰⁸

[CodeGen](#) ²¹¹

[Validation](#) ²¹³

[Testing](#) ²¹⁵

[SynEdit Downloads](#) ²¹⁸

8.1 Getting started

Prerequisites

- Delphi / RAD Studio installed
- Codolex installed

Set up project

To add an activity to Codolex, we need to create a so-called plugin that will add the necessary components and code. Basically, this is a BPL. We created a plugin template for you to download, so you don't need to start from scratch. Click the following link to download the zip file with the package source code:

[GDK-codolex-plugin-template.zip](#)

Project settings

Once you downloaded the project, we need to take a look at the settings to make it ready for the new activity.

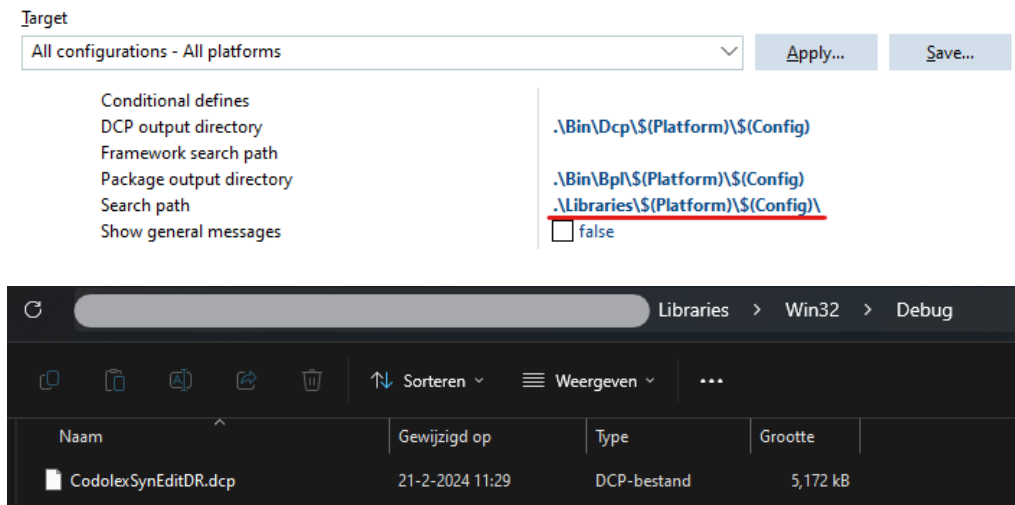
Add SynEdit

The project makes use of the installed Codolex files. The only thing you need in addition is the CodolexSynEditDR.dcp file for compiling. If you run the project straight after installation the following error will occur:

[dcc32 Fatal Error] CodolexComponents.Template.Core.dpk(34): E2202 Required package 'CodolexSynEditDR' not found

Download the [SynEdit](#) ²¹⁸ component and place the component in your search path. Be sure to get the right version for Codolex and your RAD Studio version. In the example, we will use Codolex 1.7.0 and RAD Studio 11.3.

You need to include the path to this file in your search path.



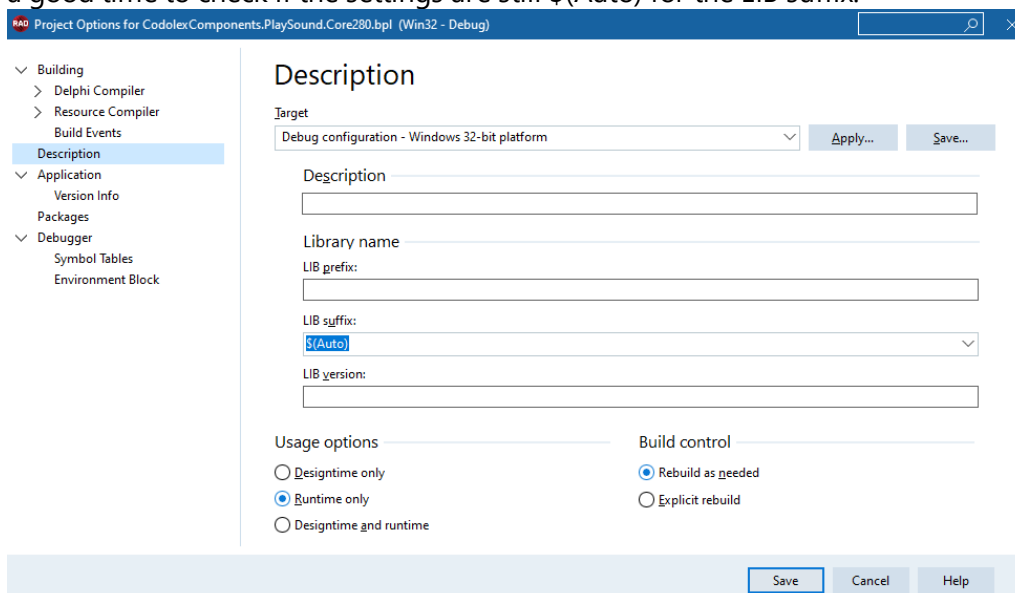
Rename template files

Open the template project and rename the template part in the file "CodolexComponents.Template.Plugin.pas" to something that defines your plugin. Let's name it CodolexComponents.PlaySound.Plugin.pas. Don't forget to change the name defined in the PluginName function.

The '**Plugins\Core\Example**' folder also contains the files with the name example in them. Every activity you develop should have this file set. In the implementation part, we will take a closer look at each one of these files. For now, you only have to rename the word Example to the name of your plugin.

Set project BPL name

In addition to renaming the files, it's good to rename the BPL project. This is also a good time to check if the settings are still \$(Auto) for the LIB suffix.



This ensures that the right suffix for the plugin is used. So, if you installed Codolex for Delphi 11.3, and you are building the plugin in the same version, the suffix 280.bpl is used. If the BPL does not have the right suffix, it will not be used by Codolex.

Define your activities

In the Components function in the Codolex.YourProjectName.Plugin.Pas file, you need to define the activities your plugin should contain. If you only want one activity, you can change the name of TExamplePluggableComponent to your class. If you want more activities, be sure to create them with the right names. In this example we've looked at two activities needed. Let's name them "TPlaySoundPluggableComponent" and "TStopSoundPluggableComponent"

```
function TCodolexPlaySoundPlugin.Components: IList<IFlowPluggableComponent>;
begin
    Result := TCollections.CreateList<IFlowPluggableComponent>;

    var PlaySoundActivity := TPlaySoundPluggableComponent.Create;
    Result.Add(PlaySoundActivity);

    var StopSoundActivity := TStopSoundPluggableComponent.Create;
    Result.Add(StopSoundActivity);
end;
```

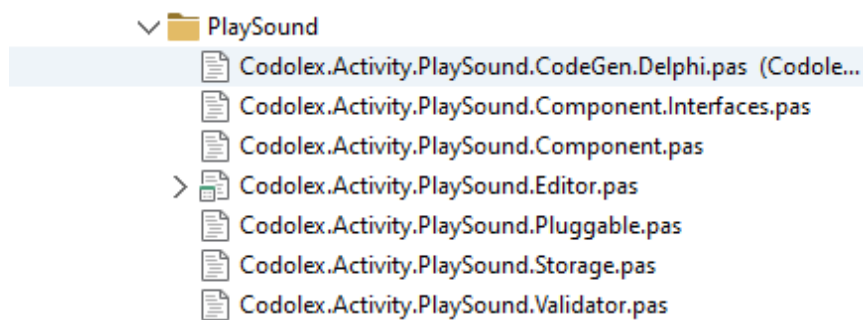
These are the activities that will show up in the activity palette once the plugin is complete and installed. But, as you can see from the error lines, we need to implement these activities.

8.2 Implementation

For each activity, we need to implement the following files:

- Codolex.Activity.YourActivityName.Pluggable.pas
- Codolex.Activity.YourActivityName.Component.Interfaces.pas
- Codolex.Activity.YourActivityName.Component.pas
- Codolex.Activity.YourActivityName.Editor.pas
- Codolex.Activity.YourActivityName.Editor.dfm
- Codolex.Activity.YourActivityName.Storage.pas
- Codolex.Activity.YourActivityName.Validator.pas
- Codolex.Activity.YourActivityName.CodeGen.Delphi.pas

In the example folder, there is an example for every listed file. Remove the example files from the project to prevent errors when adding your activity files. To start, we can copy these files into a new folder, rename them, and add them to the project. The easiest way is to rename every instance of example in the names and contents to your activity name. Continuing with our example, let's name this "Playsound".



After we added the files for PlaySound, we can look at the files one by one.

Pluggable

This class is the base for your activity and connects all the needed units. In this class, you can provide the details for the activity, like the name and description. The pluggable class defined here is added to the list of activities, supported by the plugin, in the

"CodolexComponents.YourActivityName.Plugin.pas".

The declaration should look something like this:

```
TPlaySoundPluggableComponent = class(TInterfacedObject,
IFlowPluggableComponent, IFlowPluggableComponentWithValidation,
IFlowPluggableComponentWithEditor)
public
    function Name: string;
    function Description: string;
    function Category: string;

    function CreateComponent(const Flow: IFlow): IFlowComponent;

    function CodeGenerator(const FlowComponent: IFlowComponent):
IFlowPluggableComponentCodeGen;
    function Storage(const Container: TContainer):
IFlowPluggableComponentStorage;
    function ShowEditor(const FlowComponent: IFlowComponent; const
Container: TContainer): TModalResult;

    function Validator(const FlowComponent: IFlowComponent):
ICodolexValidator;
    function CreateEditor: IFlowPluggableComponentEditor;

    function SupportedLanguages: TCodolexLanguages;
end;
```

Let's go over a few functions to clarify what they are needed for.

- Name: Provides the name for the validator and the storage.
- Description: Provides the default description for your activity. This is how the activity will be shown in the palette.

- **Category:** Defines the category the activity can be found in in the palette. It's recommended to provide a unique name that does not conflict with the other Codolex categories. Keep them the same for the activities you create here. For both activities, let's use the Playsound category.
- **SupportedLanguages:** This returns a list of languages that this activity is going to support. For now, Delphi is the only language used. When more languages are supported by Codolex, be sure to update this section when you want to use it for another language. The available languages are specified in the 'TCodolexLanguage' Enum from the 'CodeGen.Language.Defines' unit.
- **GetTags:** This function is not present in the example yet but could be a great function to add. If your activity has multiple options, and you know a few of these options are going to be used quite often, you can add a tag for each of these options to make them faster to select.
- **Other functions of the pluggable component:** The other functions are needed for Codolex to create the instance and storage files etc. Assuming you copied them correctly, these don't need to be changed.

Component interface

To create your activity you need a component. This of course starts with creating the interface. Your interface has to be inherited from the 'IFlowActivity' interface. In your interface, you can specify the properties you need for your activity.

The first thing you need to resolve here is the GUID, Every activity needs its own GUID, which you can create via the shortcut **ctrl+shift+g**. Replace the {\$MESSAGE WARN 'Set GUID'} with a unique GUID.

In our case, we need the user to provide a file for the sound and the mode for the Playsound function (SND_NODEFAULT Or SND_ASYNC Or SND_LOOP). So we can define a property SoundFile of the type IVariable and PlayMode of the type String

```
type
  IFlowActivityPlaySound = interface(IFlowActivity)
    [ '{344D0F87-1DB3-4C89-B5E6-57A638C649E9}' ]

    function GetSoundFile: IVariable;
    function GetPlayMode: string;

    procedure SetSoundFile(const Value: IVariable);
    procedure SetPlayMode(const Value: string);

    property SoundFile: IVariable read GetSoundFile write
    SetSoundFile;
    property PlayMode: string read GetPlayMode write SetPlayMode;
end;
```

Component Implementation

When you've finished your interface, you also need to make its implementation. This is done by creating a class that inherits from the 'TFlowActivity' class, as well as the 'IFlowComponent' interface and your newly created interface.

Implement the property functions. The complex fields like IVariable should be ignored by the JSONMarshaller in the storage because we need to serialize the references ourselves, so we add the [JSONMarshaled(False)] directive before the variable declaration. Add the REST.Json.Types unit to the interface uses section as well. Otherwise, this directive gets ignored, and you will get errors.

```
uses
...
REST.Json.Types,
...;

TFlowActivityPlaySound = class(TFlowActivity, IFlowComponent,
IFlowActivityPlaySound)
const
  FQName = 'Template.Core.PlaySound';
  DescriptionOfComponent = 'Play sound';
private
  [JSONMarshaled(False)]
  FSoundFile: IVariable;

  FPlayMode: string;
...
```

Next to the properties, you can also see the 2 constants. FQName and DescriptionOfComponent. These properties define the name and description of your component. The name is the same as described in the Pluggable section above. The description is used in your flow as the name of the activity.

There are also some procedures which are used by Codolex. The first is InitComponent. In this procedure, you set the name, description, and icon of your component. This is done by setting the variables of the TFlowActivity we inherited in the class.

For the Icon, you can use any UTF-8 character and color. Note that Delphi is not able to display all UTF-8 characters and might show as an invalid character in the IDE. You can also provide a color to use in the activity. It's recommended to keep them the same for all the activities you add in one component.

```
procedure TFlowActivityPlaySound.InitComponent;
begin
  inherited;
```

```

FComponentName := FQName;
FComponentDescription := DescriptionOfComponent;

Icon.Init('▶', clWebLimeGreen);
end;

```

In the init component function, it's also possible to add default values to the properties defined in the interface.

```

procedure TFlowActivityPlaySound.InitComponent;
begin
    ...
    FPlayMode := 'Async';
    ...
end;

```

With the inherited function InitializeElement, you can specify a return variable. If you would like this variable to be of a custom type, then use the next piece of code

```

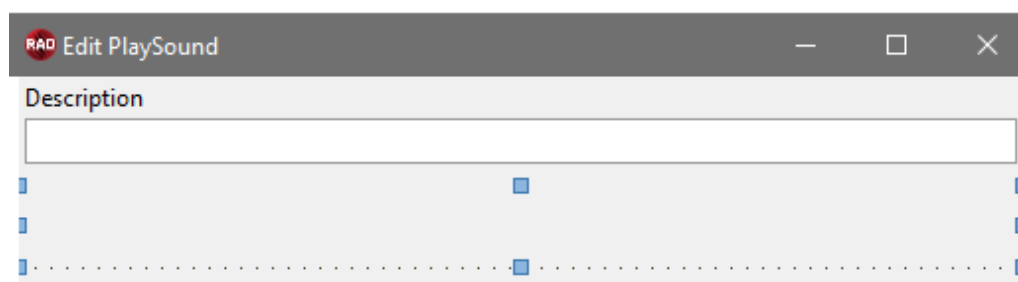
procedure TFlowActivityPlaySound.InitializeElement;
begin
    inherited;

    _ReturnVariable.VariableType := TDataType.Custom_;
    _ReturnVariable.CustomTypeName := 'TYourCustomType';
end;

```

Editor

Next, continuing with our example, the editor file will be the property screen for this activity. This is the screen you are going to see in Codolox when editing an activity. Here you can define how the user can provide input for the properties. You can design this screen the way you like it, or, the way you think will be the most intuitive for the user of the activity. For ordinal types, you can use the default input fields that Delphi has to offer. For example, if you want the user to fill in a name, just put a TEdit on the form with a label that clarifies that is the name input. And just like that, the description input that is already present.



Another option is to use the editor's inputs provided by Codolex, since they are needed for complex properties like the `IVariable`. These editors are unavailable during design time, so you must initialize them at runtime. This can be done in the initialize section.

But before we can initialize them, we need to define a few things.

1. Add these uses for your input to the interface section. (if not there already)

```
Codolex.Activity.SoundFile.Component.Interfaces,  
Codolex.Modelling.Helper.Interfaces,  
ProjectModel.Variable.Interfaces,  
Codolex.Editors.PropertyInput.FlowVariable;
```

To see the other editors that are available, use the autocomplete on "Codolex.Editors.PropertyInput." this will provide you with options for entity selection, lookups, etc.

If the editor does not see the available classes. Try to compile the project, and restart so load the configuration correctly

2. Create class properties for the `IVariable` and the `FlowVariable` input.

```
Private  
    FSoundFile :IVariable;  
    FSoundFileInput: TfrmFlowVariableInput;  
    ...
```

3. Add a resource string for the label name. This is not mandatory but could make it easier to change language settings if needed.

```
resourcestring  
    SoundFileInputLabel = 'Sound file (.wav)';
```

4. Place a panel in the designer where you want the input to be. This panel is going to be filled with the input with `alClient` alignment. It's recommended to give this panel the `alTop` align setting, as seen in the picture above.

5. Add the "OnChange" function for the sound file. This function can be assigned to the "OnChange" event after adding the input, to save the value after the user provides input.

```
procedure OnSoundFileChange(Variable: IVariable);  
    ...  
procedure TfrmActivityPlaySound.OnSoundFileChange(Variable:  
    IVariable);  
begin
```

```
FSoundFile := Variable;
end;
```

6. Add the CanChooseVariable function for the sound file input. This function also needs to be assigned after the input is created. In this case, it's needed to provide the right suggestions for the user in the input field. Here you can also limit the options for the user. For example, we want the user to select a string for the sound file.

```
function
TfrmActivityPlaySound.CanChooseVariableForSoundFile(Variable:
IVariable): Boolean;
begin
    Result :=
TFlowVariableDefines.IsVariableCurrentSelected(Variable,
FSoundFile) or
    TFlowVariableDefines.IsVariableString(Variable);
end;
```

The "TFlowVariableDefines" class can be used for all sorts of helpers. Here is how you can check for a custom type:

```
function
TfrmActivityPlaySound.CanChooseVariableForSoundFile(Variable:
IVariable): Boolean;
begin
    Result := False;

    if Assigned(Variable) then
    begin
        var IsCustom := Variable.VariableType = TDataType.Custom_;
        if IsCustom then
        begin
            var CustomType := Variable.CustomTypeName.ToLower;
            Result := CustomType = 'Array of const';
        end;
    end;
end;
```

7. Now we can go on with the initialization code.

```
var FSoundFileInput := TfrmFlowVariableInput.AddInput(
    pnlSoundFile,
    FFlowModellingHelper,
    FlowComponent,
    FSoundFile
);
```

We can use the AddInput function from the Input form class to create the component and add in the class property we created in step 1. By providing the panel, the class automatically adds the component as a child of the panel.

After we have created the input, we have to add some properties and events and initialize the component.

```
FSoundFileInput.LabelName := SoundFileInputLabel;  
FSoundFileInput.InitializeInput;  
FSoundFileInput.OnChange := OnSoundFileChange;  
FSoundFileInput.OnFilter := CanChooseVariableForSoundFile;  
FSoundFileInput.LoadVariableSelection;
```

Repeat these steps for all your properties to make every property available in the editor. When all the editors are set up, the user can edit the value of the input. But, for the changes to be persistent, we need to save the values in the component as well.

This can be done with the function `SaveToModel`.

```
procedure TfrmActivityPlaySound.SaveToModel;  
begin  
    FActivity.SoundFile := FSoundFile;  
    FActivity.Description := editDescription.Text;  
    FActivity.PlayMode := FPlayMode;  
end;
```

Be sure to fill in the contents of this function to save all your properties.

Storage

The storage unit describes how your activity should be saved to and read from the JSON. This is done using 3 classes. Storage, Serializer, and Deserializer. We don't need to worry about the storage class. This class is here to provide an instance of the serializer and deserializer.

The serializer needs a `DoSerialize` function that serializes the references for us that we did mark as "JSONMarshaled False".

```
procedure TActivityPlaySoundSerializer.DoAfterSerialize;  
var  
    Activity: IFlowActivityPlaySound;  
begin  
    inherited;  
  
    Activity := FEntity;  
    SerializeReference<IVariable>(FJson, 'SoundFile',  
    Activity.SoundFile);  
end;
```

We need to deserialize the same references in the afterparse.

```
procedure TActivityPlaySoundDeserializer.DoAfterParse;
```

```

var
  Activity: IFlowActivityPlaySound;
begin
  inherited;

  Activity := FEntity;
  DeserializeReference(FJson, 'SoundFile', procedure (const
    Variable: IVariable)
  begin
    Activity.SoundFile := Variable;
  end);
end;

```

Remember, these functions need to be overridden in the declaration to make use of the inherited function.

8.3 Tags

On the [Getting started](#) ¹⁹⁷ page, while discussing the Codalex.Activity.PlaySound.Pluggable.pas File, we mentioned something about implementing tags. Now that we have our component defined, we can add tags to complete the plugin definition.

An example of this can be found in the create variable activity of Codalex. This activity can also be found in the palette by searching for boolean. When you drag the activity into the flow while searching for "boolean", the variable type boolean is automatically selected.

We could implement this in our plugin for the Playmode selection.

We need to add **IFlowPluggableComponentWithTags** to the pluggable class interfaces, and add the functions GetTags and Initialize.

```

function GetTags: TArray<string>;
procedure Initialize(const FlowComponent: IFlowComponent; const
  WithTag: string);

```

In the GetTags function, we can define a string array that contains all the tags the activity can be found on.

```

function TPlaySoundPluggableComponent.GetTags: TArray<string>;
begin
  Result := ['Async', 'Loop'];
end;

```

In the Initialize procedure, we need to set the value based on the selected tag. The WithTag parameter contains the selected tag. We can check if that tag was async or loop, and then assign the correct playmode.

```

procedure TPlaySoundPluggableComponent.Initialize(const
FlowComponent: IFlowComponent; const WithTag: string);
var
    Activity: IFlowActivityPlaySound;
begin
    if not Supports(FlowComponent, IFlowActivityPlaySound, Activity)
then
        raise
EInvalidOpException.CreateFmt(ValidationNotAvailableForActivity,
[FlowComponent.ComponentName]);
    if WithTag = 'Loop' then
        Activity.PlayMode := TPlayMode.SND_LOOP
    else if WithTag = 'Async' then
        Activity.PlayMode := TPlayMode.SND_ASYNC;
end;

```

With this, it is easier for the user of this activity to directly select the right Playmode.

8.4 Defined entity

When creating an activity, it could be possible that a new entity for the properties, or result variable is needed.

It's possible to add an extra element to the [Plugin datasources](#) ⁵⁵ within the component.

Let's say we want to provide an extra option in our activity to provide an entity with the filename and play mode as attributes.

We need to take the following steps

1. Define guid(s) for entities to add
2. Add "Codolex.Activity.YourActivityName.DefinedEntity.pas".
3. Add entities to generate action to the '...Component.Pas'.
4. Call entities to add in the '...Puggable.pas'.
5. Extra info

1. Define guid(s) for entities to add

Go to the 'Component.Interfaces.pas' file, and add a **const** section to the interfaces. Use Ctrl+Shift+G to generate a guid for every entity

```

interface

uses
    ...;

const
    PlaySoundConfigGuid = '{55FC2C2E-AB35-4BD2-ABC4-F6950FE10A0D}';

```

2. Add DefinedEntities.pas

Add a new unit, and use the following code in the unit (Replace playsound with your activity name)

```
unit Codolex.Activity.PlaySound.DefinedEntities;

interface

uses
  DataModel.Entity.Interfaces,
  Codolex.Spring.Collections;

type
  TCreateDataEntityFunc = reference to function(const Name:
string): IDataEntity;

  TPlaySoundDefinedEntities = class
  private
    class function CreatePlaySoundConfig(const CreateEntity:
TCreateDataEntityFunc): IDataEntity;

  public
    class procedure Get(const Entities: IDictionary<string,
IDataEntity>; const CreateEntity: TCreateDataEntityFunc);
    end;

  implementation

  uses
    DataModel.DataTypes.Defines;

  { TPlaySoundDefinedEntities }

  class procedure TPlaySoundDefinedEntities.Get(const Entities:
IDictionary<string, IDataEntity>; const CreateEntity:
TCreateDataEntityFunc);
  begin
    var PlaySoundConfig := CreatePlaySoundConfig(CreateEntity);
    Entities.Add(PlaySoundConfig.Name, PlaySoundConfig);

    //repeat this for as many entities as you need
  end;

  class function
  TPlaySoundDefinedEntities.CreatePlaySoundConfig(const CreateEntity:
TCreateDataEntityFunc): IDataEntity;
  begin
    Result := CreateEntity('PlaySoundConfig');
    Result.Guid := PlaySoundConfigGuid;

    Result.AddStringField('FileName', '{4061BAB6-EE7E-4D62-B78D-
33A07166C4B6}');
```

```

    Result.AddStringField('PlayMode', '{1DBACDBA-5A4B-4072-A80-
2FE703194F01}');
end;

end.

```

3. Add entities to generate action to the '...Component.Pas'.

In the **Component.pas** add the following files to the interface uses:

- DataModel.Entity.Interfaces,
- Codolx.Spring.Collections,

Add the

Add the following class procedure to the protected functions

```

class procedure DefineEntities(const Entities: IDictionary<string,
IDataEntity>); override;

```

Add the created unit 'Codolx.Activity.PlaySound.DefinedEntities;' to the implementation uses section.

This makes it possible to call the class function **Get** From the unit.

```

class procedure DefineEntities(const Entities: IDictionary<string,
IDataEntity>); override;
begin
    inherited;
    TPlaySoundDefinedEntities.Get(Entities, CreateDataEntity);
end;

```

4. Call entities to add in the '...Puggable.pas'.

In the **Pluggable.pas** add the following files to the interface uses:

- DataModel.Entity.Interfaces,
- Codolx.Spring.Collections,

Add the interface **IFlowComponentWithDataEntities** to the list of used interfaces for the plugin object.

Add the following class procedure to the public functions

```

function GetDataEntities: IReadOnlyCollection<IDataEntity>;

```

Implement the function by calling the **GetDefinedEntities** function from the component

```

function TPlaySoundPluggableComponent.GetDataEntities:
IReadOnlyCollection<IDataEntity>;
begin
    Result := TFlowActivityPlaySound.GetDefinedEntities.Values;

```

```
end;
```

This ensures the entities are created when the plugin is loaded into codolex.

5. Extra info

It's possible to add multiple entities this way, and it's also possible to add associations between those entities.

To add an association, use the following code:

```
...
var Association :=
ParentEntity.AddAssociationAsParent(ChildEntity);

Association.Guid := //Generate a guid;
Association.AssociationType := TAssociationType.OneToMany; //one
to one is also possible
Association.ChildName := 'ChildNameAssociation';
Association.Name := 'AssiciationName';
...
```

8.5 CodeGen

Now comes the most important part: code generation.

In the CodeGen file, you can find an empty procedure code that has the parameter `SourceCode`. This is a function from the interface and will be used in the code generation for a flow when the activity is used. Use this function to add lines to the generated code. This can be done by the `add` function on the `SourceCode` object. For example, hello world:

```
procedure TFlowActivityExampleDelphiSource.Code(const SourceCode: ISo
begin
    Sourcecode.Add('Hello world!');
end;
```

The source code object has multiple functions that can help with designing the code. We will cover a few functions that will be used in most activities. Keep in mind that every function in the `SourceCode` object returns the `SourceCode` to be used in the next line. That enables you to chain the `SourceCode` functions to make it easier and more readable.

.Add

The `add` function adds lines to the code. To start a new line, call another `add` function. The contents can be a string or an array of strings.

.IncreaseIndent and .DecreaseIndent

When designing the generated code, it's a good idea to keep in mind that it should be readable. Thus, having proper indentation is very important. Thankfully,

Codolox uses `IncreaseIndent/DecreaseIndent` to place the code with spaces before it in the result. The `SourceCode` remembers the last indent, so, for a function or an if-statement where multiple lines need to have a greater indent, use `IncreaseIndent` once before, and `Decrease Indent` once after.

.Begin_ and .End_

This is useful for creating pieces of code that need a begin and end, like if-statements and loops, but also for functions and procedures. These two functions simply place a beginning and end tags to your code.

The `.Begin_` function automatically increases the indent in the code and the `.End_` function automatically decreases it.

```
SourceCode
  .Add('if not ::SoundFile.IsEmpty then')
  .Begin_
  .Add('sndPlaySound(::SoundFile, ::PlayMode);')
  .End_;
```

will result in

```
if not Soundpath.IsEmpty then
begin
  PlaySound(Soundpath, 0, SND_NODEFAULT);
end;
```

.Param

To help with designing the code, you can define parameters that the `SourceCode` object can parse when creating the result. Those parameters can be used in the add function with double points. "::", as seen above for the `SoundFile` and `PlayMode` parameters.

```
SourceCode
  .Param('SoundFile', FActivity.SoundFile.Name)
  .Param('PlayMode', TRttiEnumerationType.GetName(FActivity.PlayMode))
  .Add('sndPlaySound(::SoundFile, 0, ::PlayMode);');
```

will result in

```
sndPlaySound(Soundpath, SND_NODEFAULT);
```

Given that `FActivity.SoundFile.Name = 'SoundPath'` and `FActivity.PlayMode = 'SND_NODEFAULT'`.

.Variable

Sometimes you might need a new variable in your generated code. This can be added with the `.Variable` function. Provide a name and a datatype. Optionally, provide a dependency when you need to import a unit for the datatype.

```
.Variable('TempString', 'string');
```

will result in

```
var TempString: string;
```

There is also an option to declare the param, so that you can use the variable directly in your code.

```
.VariableWithParam('TempParamName', 'TempEnumVar', 'TMyOwnEnum', 'Unit.o
...
.add('::TempParamName');
```

will result in

```
uses
  Unit.of.enum
...
var TempEnumVar: TMyOwnEnum;
```

on the place where the param was added.

.AddDependency

Speaking of dependencies, you can also use the .AddDependency function to add "Unit.of.enum". By default, this use is added in the implementation section. If you want it in the interface, use:

```
.AddDependency('Unit.of.Enum', TUsesSection.Implementation_)
```

While there are more functions and variations of the mentioned function in the SourceCode object, you can always find them in the auto-complete, and just try them out with a few example projects. For our activity, this is enough for now.

```
SourceCode
.AddDependency('MMSystem', TUsesSection.Interface_)
.Param('SoundFile', FActivity.SoundFile.Name)
.Param('PlayMode', TRttiEnumerationType.GetName(FActivity.PlayMode)
.Add('if not (::SoundFile = '') then')
.Begin_
.VariableWithParam('SoundFileChar', 'SoundFileWideChar', 'PWideCh
.Add('::SoundFileChar')
.add('::SoundFileChar := PWideChar(::SoundFile);')

.Add('sndPlaySound(::SoundFileChar, ::PlayMode);')
.end_;
```

8.6 Validation

You can see in the generated code above that the assumption is made that a SoundFile object is always available. The user of the activity has to provide this

file, but it may very well be that the user forgets to provide one, or deletes the variable that was being used.

To prevent errors in that case, we need to add some validation. We are going to validate if the soundfile variable is selected in the activity. This can be done in the `Codalex.Activity.YourActivityName.Validator` file. This file provides a `DoValidate` function to Codalex to execute when the validation is called. This function has the `Variable FElement` available which will be the instance of your component when validating. You can use this element to perform all kinds of validations, like if a property in the component is filled, when working with numbers in inputs, if the numbers are correct and within limits, etc.

In the example, an error is thrown when the validator is not changed. However, let's now change the validator to remove the error. Let's also check if the `SoundFile` property is filled, so we know the generated code won't give an error.

```
resourcestring
    NoSoundFileProvided = 'No sound file provided for %s';

procedure TFlowActivityPlaySoundValidator.DoValidate;
begin
    inherited;
    if not Assigned(FElement.SoundFile) then
        AddError(NoSoundFileProvided, [FElement.ComponentName], 'SoundFile');
end;
```

.AddError

When the property is not filled, we can add an error to the validator with the "AddError" function. This function can make use of the format functionality. Be sure to also provide the name of the property that the error is for. This is not used for now but can be used in the future to show more specific error messages or show them in the right place in the editor.

When an error is added for a component, the code generation will not execute, and an error is shown at the place where the code should be.

.AddWarning

The `AddWarning` function can also be used to let the user of the activity know that something might not be right or could be improved. For now, this will also prevent the code from generation, but this could be removed in the future.

```
var Soundpath: string;
Soundpath := 'file';
{$MESSAGE WARN '"Play sound": No sound file provided for Template.ComponentName'}
```

8.7 Testing

When adding code, it's always important to use unit tests. With the example project, a unit test and integration test project are provided. We will use both projects. The unit test to test the validator, and the integration to check if the generated code can be compiled.

Unit test

The unit test project also has an example file that we need to remove, copy and add to the project.

- "Codolex.Activity.YourActivityName.Tests.pas"

Rename this file the same way as with the components files.

The test project uses the DUnitX TestFramework to perform the unit tests.

Assuming there was no problem with copying and adding the new files, we don't need to worry about how this works.

By default, three things are tested in the unit test

1. If the creation of the component/activity does not raise an error.
2. If the codegen does not raise an error when called.
3. If the validation does not raise an error by default.

It's recommended to keep these tests in the project, but also expand on them. In our project, we're going to check if the validator indeed does raise an error if there is no SoundFile.

To add some extra tests, add a procedure with the [Test] directive

```
...
public
[Setup]
procedure Setup; override;

[Test]
procedure TestCreateComponent;

[Test]
procedure TestCodeGenAvailable;

[Test]
procedure TestValidatorAvailable;

[Test]
procedure Validation;
end;
```

This procedure will be executed when the tests are executed.

In this article, we won't dive too deep into the options of the test and validators, but a good place to start is to copy this line to use the validator. The .Validator function at the end of the line expects an activity, this is the activity we just

created to test, but can be any activity. You can also add multiple activities to test in combination.

```
var Validator := (FPluggable as  
IFlowPluggableComponentWithValidation).Validator(SoundFileActivity)  
;
```

The first thing to test is if the validator does indeed return an error when no SoundFile is provided

```
var SoundFileActivity: IFlowActivityPlaySound;  
SoundFileActivity := TFlowActivityPlaySound.Create;  
  
var Validator := (FPluggable as  
IFlowPluggableComponentWithValidation).Validator(SoundFileActivity)  
;  
var ValidationResults := Validator.Validate;  
  
Assert.IsTrue((ValidationResults.Count = 1), 'Validation has no  
result when soundfile is empty');
```

The assert class contains many static functions to assert something with. In the example, we used it to check if there are any validation results at all, But you can also create the expected result and check if it's in the validation results with the .Contains function. Use the autocomplete to see the functions available. If something does not pass the check, an error is thrown when executing the unit tests.

```
uses  
    ProjectModel.Variable.Interfaces,  
    ProjectModel.Variable,  
    ...  
var Variable: IVariable := TVariable.Create;  
  
SoundFileActivity.SoundFile := Variable;  
ValidationResults := Validator.Validate;  
  
Assert.IsTrue((ValidationResults.Count = 0), 'Validation has error  
when component is filled correctly');
```

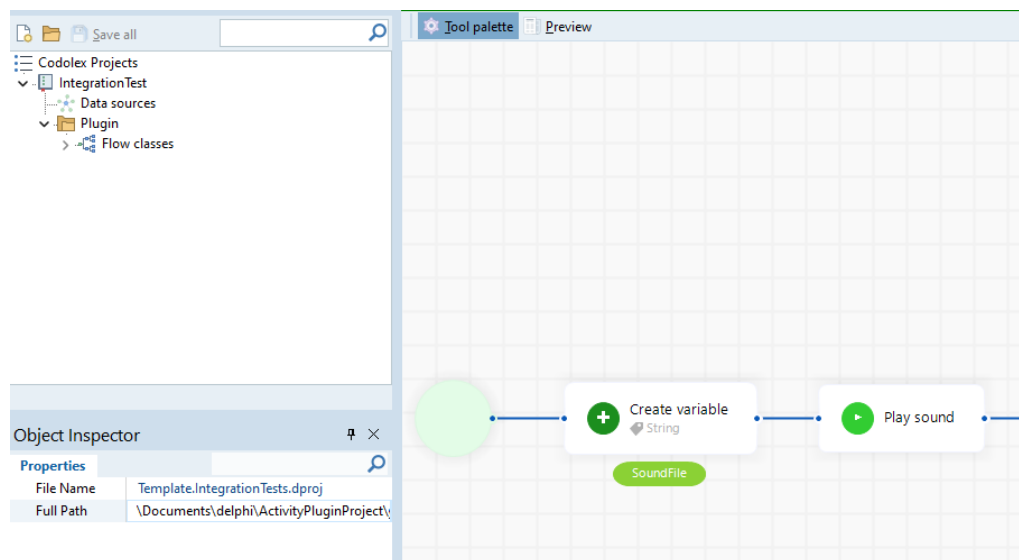
To complete the test, we're going to create a flow variable, assign it to the SoundFile, and run the validator again. After that, we can check if the result is 0.

Integration Tests

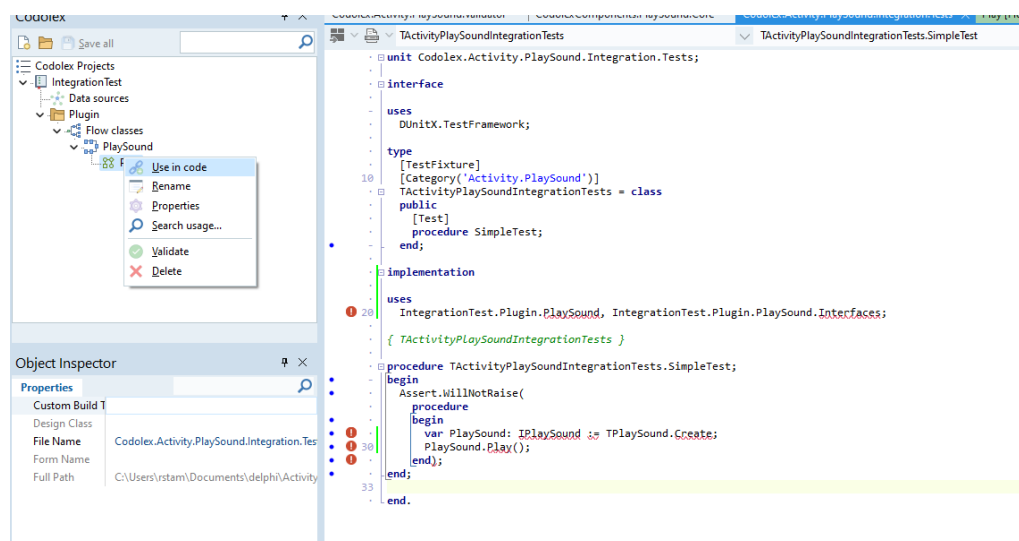
In the integration test, we are going to test the code directly with a project. Remove the example file and add a copied file for the activity as well. Select the project and open the Codolox Project Explorer. Create a new test project, or select an existing one if you already tested with a project.

Add a module, flow class, and flow to the project, and use the created activity in the flow.

If you don't find your activity in the activity palette yet, be sure that the core**0.bpl is built and copied to your appdata folder (%appdata%\Codolex\BPL). Restart Delphi if needed. Enter topic text here.



After you build your flow, you can use it in the test project with the "Use in code button" in the simple test function.



After that, you can simply run the integration test project, and see if any errors pop up.

If you are not able to build the project, something might be wrong with the generated code. If you can run the project, but the test fails, the generated code results in an error.

8.8 SynEdit Downloads

Download the synedit for your codolex version

Codolex 2.5.0: [CodolexSynEditDR2.5.0.zip](#)

Codolex 2.6.0: [CodolexSynEditDR2.6.0.zip](#)

Codolex 2.7.1: [CodolexSynEditDR2.7.1.zip](#)

Codolex 2.8.0: [CodolexSynEditDR2.8.0.zip](#)

Command line interface

9 Command line interface

It is possible to invoke code generation via the command line application. This allows Codorex to be integrated with a build server or with continuous integration. CodorexCLI.exe tool can be found in the installation directory of Codorex, and has following commands available:

- help : Show help information
- build <filepath>: Build the given Codorex project

Best practices

10 Best practices

10.1 Source control

Codolex projects have the file extension .fcp. In the location where you save the Codolex project file, a new folder with the prefix .fc will be created. This folder contains all the module and flow files. Make sure you add all the files below the new .fc folder, as they contain the logic of Codolex.

The generated source code will be stored in a new folder named .fsrc. It is a good idea to add this folder to your code repository too, as this makes it easier to compare source code updates. Do not make any manual changes to the generated files as your changes will be overwritten if you compile your Delphi project.

10.2 Useful tips and tricks

1. Use the shortcut Control+D in input fields to 'stringify' text.
2. Use the **Search usage** options to quickly navigate through your Codolex project by right clicking on entities and flows

10.3 Use multiple flows

Much like writing code, it's a good practice to keep flow short and simple. Think of a flow as a function. let the flow have 1 purpose only. If you need a flow for multiple things, think about creating multiple flow, and a new flow that calls the multiple flows. See the [Flow call](#)⁶⁷ page for more information

FAQ

11 FAQ

Codolex forum

We moved the FAQ to the Codolex forum.











Bring the forum a visit, and if you cannot find your answer, don't hesitate to ask a question...

Visit [Codolex FAQ](#) on the forum.

Release notes

12 Release notes

Find the release for your Codolex version from the following list.

[Version 2.8.0](#)  226
[Version 2.7.0](#)  230
- [Version 2.7.1](#)  231
[Version 2.6.0](#)  232
[Version 2.5.0](#)  235
[Version 2.4.0](#)  241
- [Version 2.4.1](#)  244
[Version 2.3.0](#)  244
[Version 2.2.1](#)  248
[Version 2.1.0](#)  252

12.1 Version 2.8.0

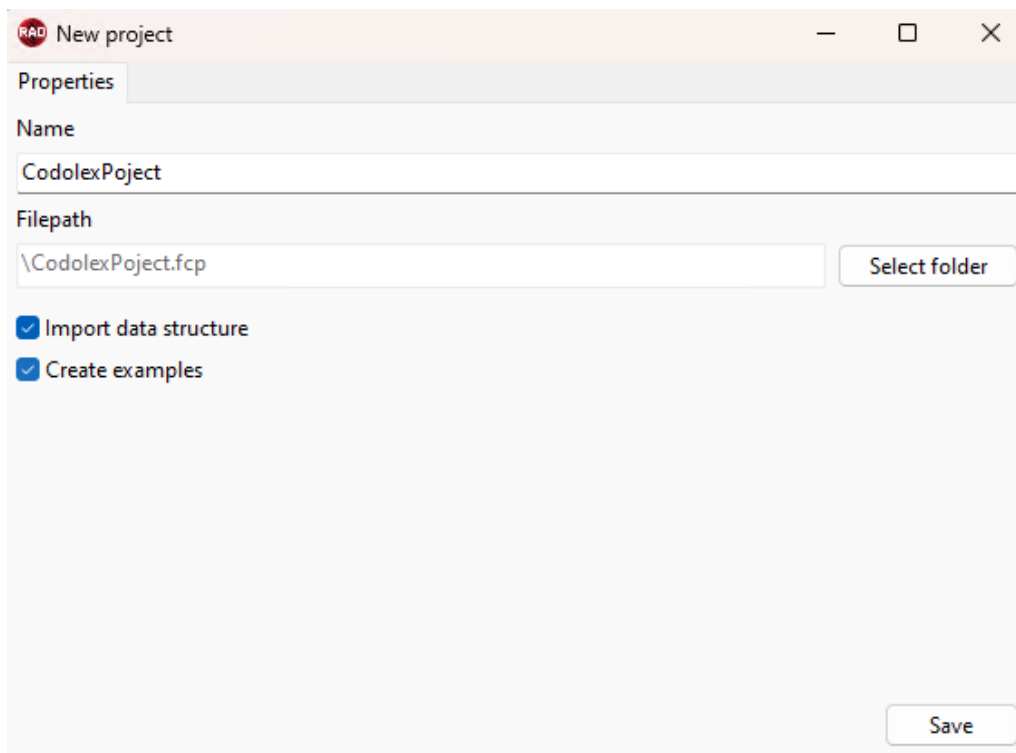
Version 2.8.0 is packed with changes, making more options available when using the Codolex API Server, and making it easier to understand Codolex with the new project wizard.

And I almost forgot to mention, bug fixes are included...

Highlights

New project wizard

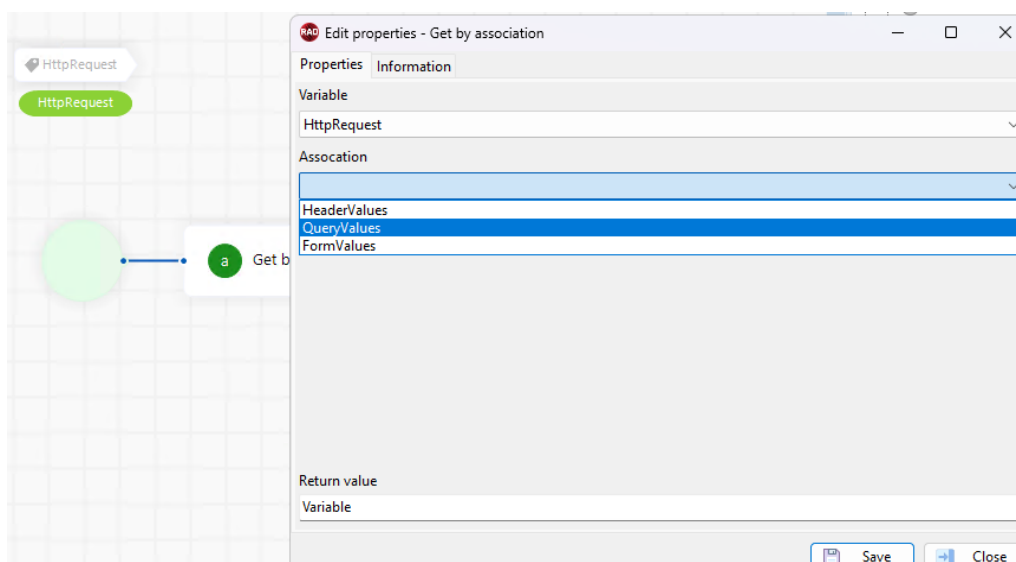
When creating a new project a wizard is shown to help you with the setup. This includes the option to select a name and location for the project and the possibility to open the new datasource wizard directly or generate some examples. This helps greatly to understand the structure of Codolex and provides some insights on how you can use it.

*New project wizard*

Query and form data parameters

When using the API server generated by Codolex, a flow that is published can use the HttpRequest parameter to receive information about the request.

In addition to the header values, we have added the association to query values and form values. This allows for more types of data retrieval from the request.

*Added query and form values*

List operation: Sort

We added the sort function to the list operation. This is possible for lists with entities, or ordinal types.

When a list is selected, it's possible to select an attribute of the list to sort on or provide a comparison with left and right.

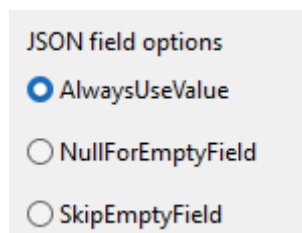
Other improvements

Basic authentication

When making a REST call with the activity in Codolex, there is an option to include basic authentication. We improved this authentication to also work when using form parameters for sending data.

JSON Export options for datasource

A datasource can now be configured to handle the export of JSON to your needs on empty values. It contains 3 options, null, 0 or '' values, or exclude from the JSON.

A screenshot of a dialog box titled "JSON field options". It contains three radio button options: "AlwaysUseValue" (which is selected), "NullForEmptyField", and "SkipEmptyField".

JSON field options

☒ AlwaysUseValue

☐ NullForEmptyField

☐ SkipEmptyField

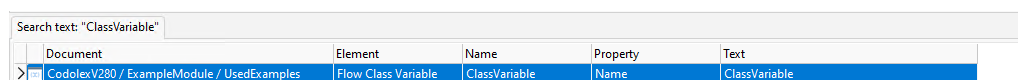
This option can be set in the 'Manage datasources' overview opened with the project explorer.

Entity with null field import

When importing entities with JSON, the entities containing 'null' values would get excluded from the import. We improved Codolex so these fields get imported as well.

Class variables text search

In the Codolex app, class variables can be found by text search. Clicking on the search result will highlight the flow class that contains the variable.

A screenshot of a search results table. The search text is "ClassVariable". The table has five columns: Document, Element, Name, Property, and Text. The first row of results shows the document "CodolexV280 / ExampleModule / UsedExamples", the element "Flow Class Variable", the name "ClassVariable", the property "Name", and the text "ClassVariable".

Search text: "ClassVariable"				
Document	Element	Name	Property	Text
> CodolexV280 / ExampleModule / UsedExamples	Flow Class Variable	ClassVariable	Name	ClassVariable

Results on search for 'ClassVariable'.

Expression editor for string change

The string split activity (along with other string activities) is given an expression editor for the input. This makes it possible to check on more combinations, like the space.

Check on value for attribute in decision activity

When an attribute from an entity is selected in a decision activity, the generated code add '.value' after the property to check on the actual value instead of the codolex field.

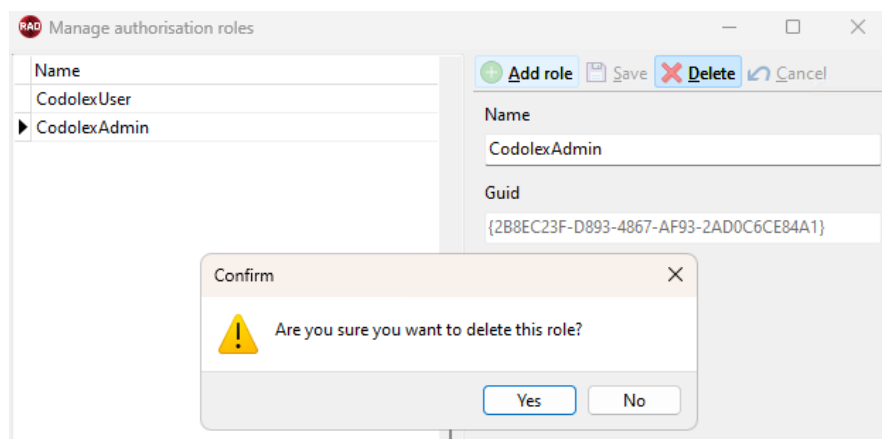
This solves issues when checking on empty.

Special characters in description

Using special characters like 'é' or 'ä' in descriptions could cause the saved description to behave strangely.

Project roles

It's also possible to delete project roles from this version and onwards. The delete checks if the role is used, and asks for confirmation to delete the role and use of it.



Delete user role.

Small improvements

- In the Codolex app, we added a menu item with a link to the webshop. Check this link for exiting components that can be added to Codolex.
- The decision expression must result in a boolean. This was not directly obvious, and is pointed out by a text hint for clarity.
- When searching for usages of a flow, the use of header flow now correctly shows in the results.
- We made the selected entity directly visible when a 'create variable' activity was configured with a list and entity.
- Deleting a flow class will not result in a save confirmation when a flow of the class is open with changes.
- The class `Codolex.Framework.Helpers.Rtti` is made available for use when creating an activity.bpl.

Bugfixes

- We added class variables as possible options to select from when using the 'Get by association' activity.
- We fixed an issue in the decision activity where selected values could be removed after an undo (Ctrl+z) action.
- We made it possible to move a loop in a loop in the editor, this got stuck on previous versions.
- We made the generated files more stable, sometimes they could get removed on reloading the project or cancelling a build. This will happen less.
- The Codolox dataset component now correctly performs an update instead of an insert when an entity is updated.
- AI Chat activity will no longer create 'Might not have been initialized' warning for the client and response variables.
- The Ctrl+enter hotkey now correctly saves the where-statement in the 'get from DB' activity.
- We fixed an issue where the autocomplete in the list operation expression editor resulted in an access violation error.
- We fixed an issue where the validation indicator would not correctly move together with the activity.

12.2 Version 2.7.0

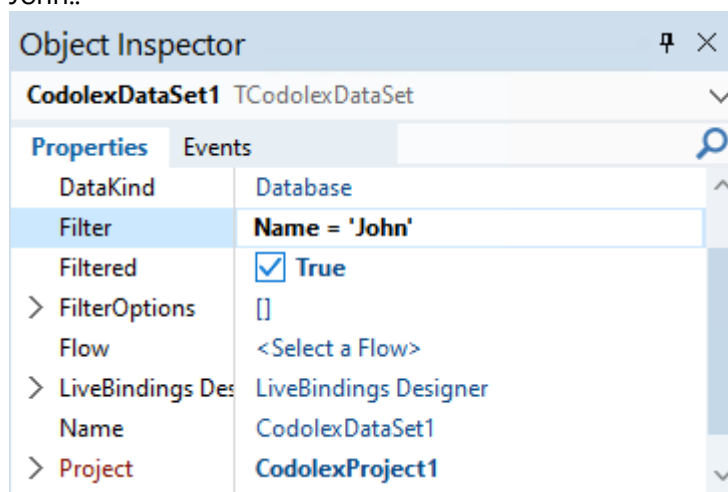
We are still working on the stability of Codolox, and that's why we want to bring you version 2.7.0. With no real highlights, but small improvements to make even more functionality possible in Codolox. And as always, fixing bugs to improve the stability.

Main Improvements

Filter on dataset

While the CodoloxDataset component did have the filter option available by default, it has not been implemented yet. So we added the implementation in this version.

That means that the following settings will result in a list of entities with the name John..




Added validation for entity import

When importing entities with attributes where the type cannot be determined of, a validation will be shown.

This will prevent the import of entities that might cause errors later on.

Other improvements

- We removed the 'Add view entity' button from a memory datasource to prevent entities with errors.
- We updated the about Codolox page to include a reference to DelphiMVCFramework, which is used to create API's
- We increased the size of the add activity button introduced in [Version 2.6.0](#) 

Bugfixes

- Defined entities in plugin activities can now have the one to one relationship.
- False conversion logs are no longer added after opening a project.
- Fixed an issue with activity size when the caption was smaller then the result data type.
- An entity used in a loop can now be used in a decision activity inside the loop.
- Improved the generation of the name of class variables created with the 'Create variable' activity.
- Removed generated by the 'Open AI Chat' activity.

12.2.1 Version 2.7.1

When testing version 2.7.0. it became clear that in some situations, the installation did leave out one or two important files. Causing the application not being able to start.

We tried to address this issue in 2.7.1, and release it instead of 2.7.0 to prevent errors on the installation of the new version.

In addition, a change made to the API is released with this version as well.

Main Improvements

Multiple custom paths for flows published as endpoint

When making a flow available as an endpoint you had the option to provide one path as a custom path. We added the option to provide multiple paths.

☒ Publish as endpoint for API

Endpoint **Authorization**

REST/HTTP method
Automatic

Custom path
It is possible to enter multiple paths. These must be separated by a semi colon (;)
directpath;lessdirectpath

Flow for response headers
HeaderFlow

Save Cancel

Separate the paths with a ';' to provide multiple paths. Both will lead to the same flow.

Bugfixes

The Application Was Unable to Start Correctly

In some cases, this error could pop up after the installation of Codolex. This could be caused by Windows Defender.

We tried to solve this issue in the new version. If this issue still arises. More information can be found on the forum:

[Codolex forum](#)

12.3 Version 2.6.0

We are happy to release the update Codolex 2.6.0. Just like the last version, we focused on improving flow handling and making the editor more reliable. Design flows even faster with this new version.

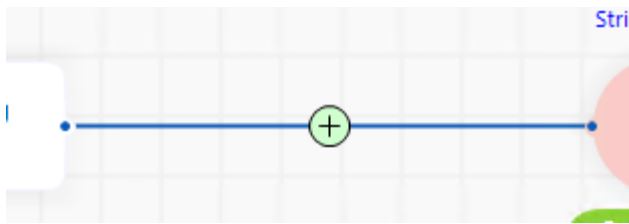
Highlights

Add activity button

We want to focus on development speed, and bringing down the amount of actions you need to get a new activity on the screen.

That's where the next button comes in.

Every sequence contains an 'Add activity' button that is visible when hovered over by the mouse.



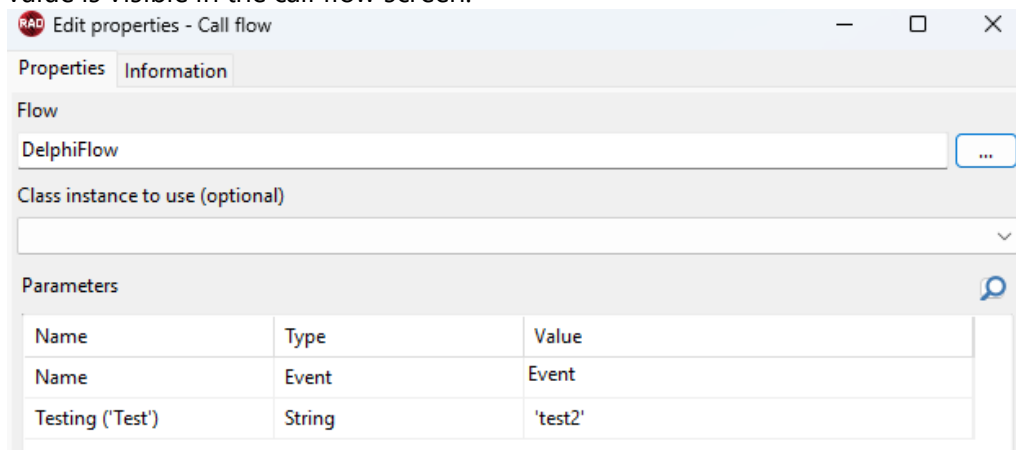
Add activity button visible on hover

Clicking this will directly take you to the add activity screen.

Other improvements

Parameter default value

A parameter can be provided with a default value in the properties. This default value is visible in the call flow screen.

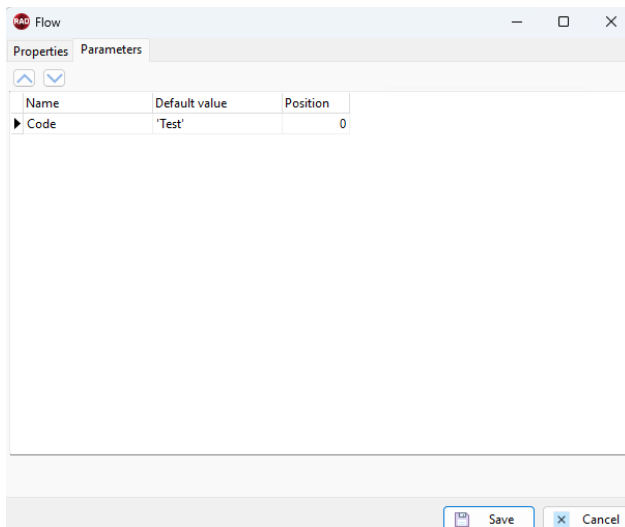


Default value 'Test' for parameter 'Testing'

Editable parameter position

Speaking about parameters, we made it possible to move the position of a parameter in a call flow action.

Using the right order for parameters can be helpful to make actions more readable.



Order of parameters in flow properties

ChatConfig for OpenAI chat

To make all the options possible when using ChatGPT, we added the **ChatConfig** entity. This entity includes all the options found in the [documentation](#). Create the entity, and fill the attributes/options that you want to use. This entity can be used in the activity to override other settings.

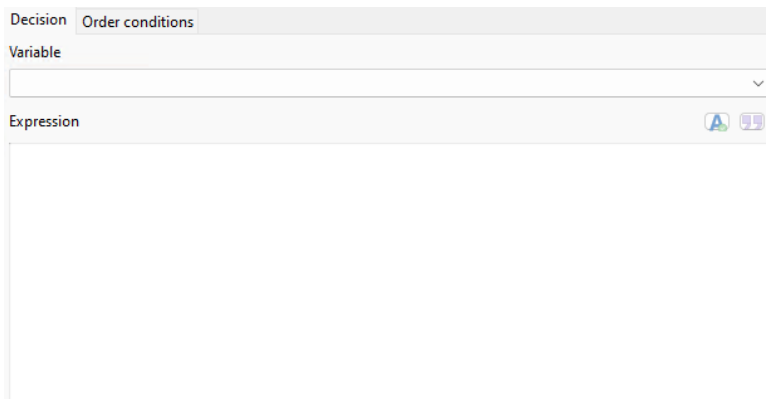
Properties	Information
Authorization key	
<input type="text"/>	
Chat configuration variable (optional)	
<input type="text"/>	
Model	
GPT 4o	
Role	

Open AI Activity

Expression editor for decision

When no variable is chosen in the decision editor, there is an expression field available.

This field can be used to make an expression that results in a boolean. e.g. `Assigned(Object)` to check if an object is assigned.



Decision expression editor

Create entity without fields

It's now possible to create an entity without fields. This is used for entities that only contains associations.

Bugfixes

- Uses for custom variables are now correctly added to the interface section when used as a return variable.
- Removed datasource type none to prevent errors.
- The API now correctly returns 0 for number attributes containing 0 instead of null.
- Fixed a bug where an incorrect suffix is used in code generation for variables.
- Fixed a bug where moving an activity inside a loop could cause it to bounce to the other side.
- The name generator for the variable no longer overwrites a name set by the user.
- Fixed an access violation when synchronizing fields on view entities without an inherited entity.
- Fixed the rendering shadow for the merge activity.

12.4 Version 2.5.0

Codolex 2.5.0 aims to improve existing components and offer stability. We added features such as the expression editor to components that did not receive the editor update earlier. Expanded on the REST activity to include pagination and resolved quite a few bugs. This all makes version 2.5.0 a must have when you look to include Codolex in your development process.

Highlights

REST Activity pagination

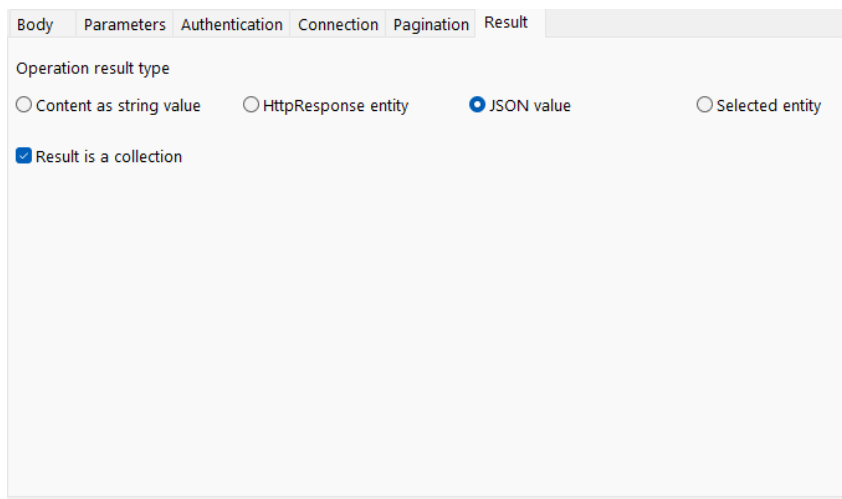
As mentioned in the intro, we added a pagination option in the REST activity. There are quite a few API's that use this structure because of large data heaps.

Use the pagination option in this case to get all the records directly into a List variable, instead of performing the call in a loop and adding the results together manually.

The screenshot shows the 'Edit properties - REST operation' dialog box with the 'Pagination' tab selected. The 'Operation' is set to 'GET' and the 'Content-Type' is 'application/json'. The 'URL' field contains 'APIUrlGetRecords'. The 'Pagination type' section has three radio buttons: 'None', 'Row/record numbers' (selected), and 'Pages'. A checkbox 'Loop until end of pages/records' is checked. The 'Request param for page size' section has a text field with 'page_size', a numeric field with '20', and a dropdown menu set to 'HEADER'. The 'Request param for row/record number' section has a text field with 'page_number', a numeric field with '0', and a dropdown menu set to 'HEADER'. The 'Response param for next number' section has a text field with 'NextPageVar' and a dropdown menu set to 'HEADER'. The 'Return the response param value in' section has a dropdown menu. The 'Return value' section has a text field with 'Response'. At the bottom right, there are 'Save' and 'Close' buttons.

You can specify in the activity where the parameters for page size and number have to go in the request, and if all objects or just this page should be returned. The latter can be useful if the API does not support a structure that the current activity can make use of, but we will include more options later on.

With this change, the result options are also expanded.



The screenshot shows a configuration window with several tabs: 'Body', 'Parameters', 'Authentication', 'Connection', 'Pagination', and 'Result'. The 'Result' tab is active. It contains the following options:

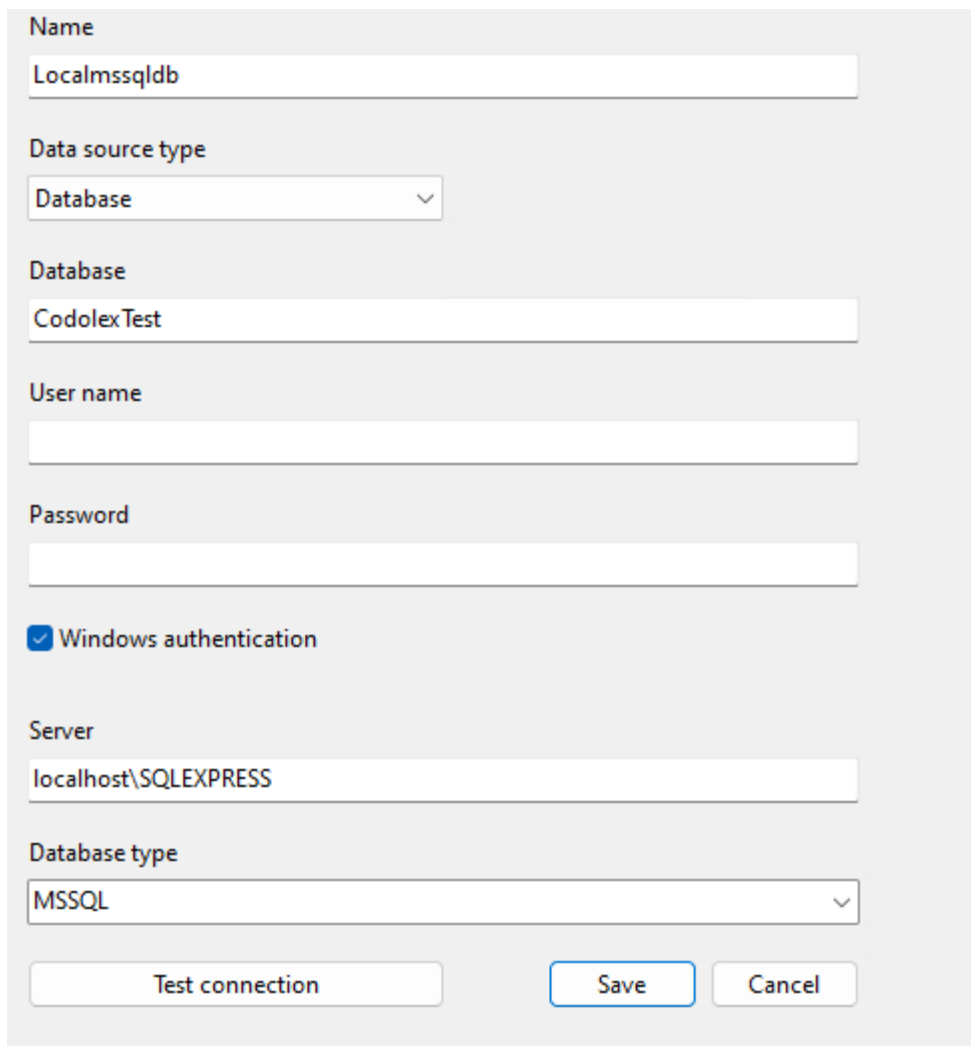
- Operation result type**
 - ☐ Content as string value
 - ☐ HttpResponseMessage entity
 - ☒ JSON value
 - ☐ Selected entity
- ☒ Result is a collection

This was needed for the pagination changes, the result of that is a collection, and that option is now available.

You can get your results in JSON or let the activity directly convert that JSON to an entity.

Windows authentication for local datasources

Some databases enable windows authentication, for example, sequel server. So to make connecting with these databases easier we added the option of using the windows authentication.



The screenshot shows a database connection configuration dialog box with the following fields and controls:

- Name:** Localmssqldb
- Data source type:** Database (dropdown menu)
- Database:** CodoloxTest
- User name:** (empty text field)
- Password:** (empty text field)
- ☒ **Windows authentication**
- Server:** localhost\SQLEXPRESS
- Database type:** MSSQL (dropdown menu)
- Buttons:** Test connection, Save, Cancel

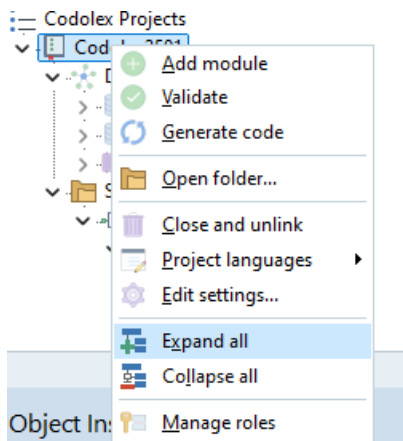
Other improvements

Api folder added to search path

The .API folder used for flows exposed as API, is now added to the search path by default. This allows you to make use of the API and start it in the same Delphi project without having to include the files yourself.

Expand/collapse in project explorer

With larger projects, the project explorer collapses the modules and flow classes by default. It's still possible to expand them all in one click, but to improve the overview of large projects, the collapse can be a useful feature.



Highlight the open flow in the explorer

The open flow in the editor is highlighted in the explorer. This improves clarity when switching between flows rapidly.

Parse collection from entity to JSON

The activity can make use of a collection of entities, and convert it to a JSON collection.

Close all in search results closes the results tab.

This works better in the codolux app vs. the IDE.

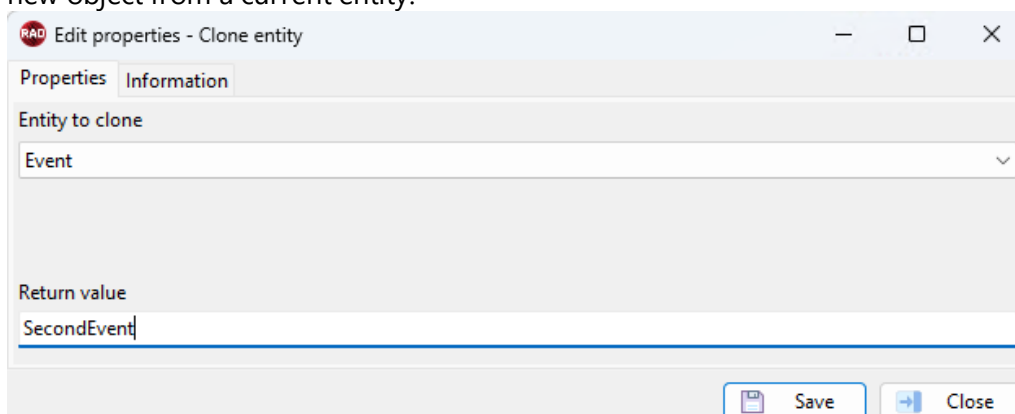
Expression added to more activities

For some activities, the editor was not yet updated to the new expression editor. For example the show dialog.

We added this to the activities where it just made sense to use the editor.

New activity: Clone entity

Instead of the existing copy attributes activity, the clone entity activity makes a new object from a current entity.



Class variable is available in more places.

In activities where a variable can be selected, the class variable is now directly available as option.

OpenAI Chat improvement

We added the option to select a role, more about roles:

<https://platform.openai.com/docs/guides/chat-completions/overview>

JSON date/time conversion

When importing or exporting data, the format `YYYY-MM-DD HH:MM:SS` (ISO 8601) is recognized and used to directly work with date time attributes.

Validations for flow names

We added better validation for flow names. Especially with duplicate names after adding or editing the name.

Adding flows with the same name or an empty name could lead to errors, so we wanted to prevent errors instead of solving them.

Validations for database file

When the database (e.g. sqlite db) is an referenced file. The local datasource connection tries to check if there is a file on the given location before trying to connect.

Updated support for TArray<T> type

Some activities needed a custom array variable to make use of an array, this results from the activity being added before the addition of the array.

We corrected the activities to offer the correct array type as possibility.

Bugfixes

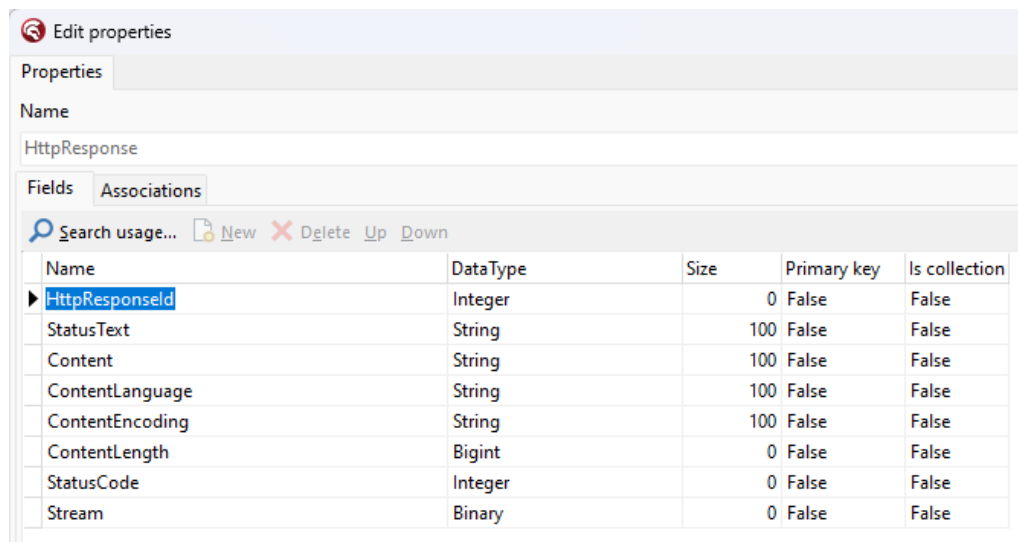
- Error message when loading a .BPL file now shows the whole bpl name.
- View entities are now always generated after the entities they are based on to prevent errors
- Copy of an activity with an exception handler no longer results in a left over sequence.
- Added a fix for the project explorer visibility in D12.1.
- The datasource view for one to many associations had the wrong direction. we changed the direction to make the view matching with the structure.
- Removed warning when using the result of execute command as flow result.
- Resolved a compile error when using a JSON entity with a parent and attribute with the same name.
- Resolved some memory leaks.
- Activity palette is now scrolled to the top instead of bottom after opening a flow.
- Using an attribute of a flow class entity in a decision does no longer result in an error.
- Used the right order for attributes in the create/change variable editor.
- Resolved an error where MVC Framework files could interfere with each other after installation.
- Fixed an issue with remainder name in the math calculation activity using integer division.

- Corrected the naming for 'squared' and 'square root' in the math calculation activity.

12.5 Version 2.4.0

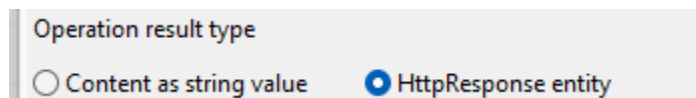
Highlights

HttpResponse for Rest activity



Name	DataType	Size	Primary key	Is collection
HttpResponseId	Integer	0	False	False
StatusText	String	100	False	False
Content	String	100	False	False
ContentLanguage	String	100	False	False
ContentEncoding	String	100	False	False
ContentLength	Bigint	0	False	False
StatusCode	Integer	0	False	False
Stream	Binary	0	False	False

The Rest activity now has an option for HttpResponse, an integrated codex entity that contains all the response details. This can be used to retrieve the status code, or the headers via the association that contains a HttpHeadersValue list.

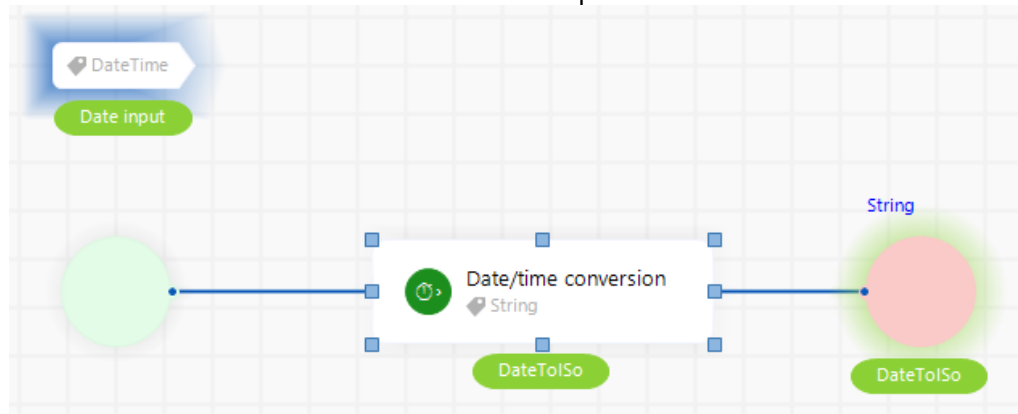


Operation result type

☐ Content as string value ☒ HttpResponse entity

Show where what is used

The flow editor can show where a variable or parameter is used or what it uses.



The selected activity is the date/time conversion.

It shows that it uses the parameter date input with the blue highlight.
It shows that the result variable DateTolso is used as return value in the flow with green highlight at the end activity.

This can greatly help in figuring out how the flow is made and will be handled.

Added options for API server

When exposing a flow as API endpoint, the server files are generated for usage in the project. The server is expanded with several options.

1. Ssl Setup

The TIdHTTPWebBrokerBridge has been made available through the property SeverInstance to add more configuration options like SSL .

2. Swagger info assignment

The swagger information can be updated for correct documentation.

```
FApiServer.WithSwaggerInfo(  
  function: TMVCSwaggerInfo  
  begin  
    Result.Title := 'TestProject API Server';  
    Result.Version := '2.4.0';  
    Result.Description := 'TestProject API Server generated by Codolex';  
    Result.ContactEmail := 'info@codolex.com';  
  end  
);
```

Updated swagger info example

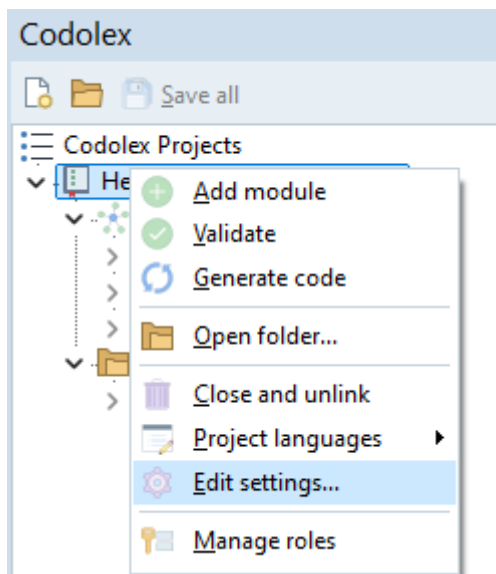
3. Added body parameters for flow endpoints

When a flow can be called with parameters, the documentation now provides info of the parameters.

4. Api settings

In the menu, an option for settings is added. These settings include only one option for now, but more will be added in the future.

The option that has been added is 'generate entities with persistent state'. This determines if the state of an entity is added in the json when an entity is directly retrieved from the database with an api call.



Features

The editor is easier to navigate

From now on, you can navigate the flow editor with right mouse click and moving the mouse.

And when moving the focus with the scroll bar, the editor moves along while scrolling. not updating after moving.

Save object can now save lists.

This saves the time of making a loop and saving the objects 1 at a time.

Other improvements

1. About information updated whit link to Codolex and GDK Software website
2. Defined entities for custom activities can aslo be defined with attributes.
3. Show dialog result is optional to prevent hints in RAD Studio.

Bugfixes

1. An issue where double round calls where done in one 'round' activity has been resolved.
2. Errors when doing a copy/paste action with a loop has been resolved.

12.5.1 Version 2.4.1

This version is a fix for version 2.4.0

Bugfixes

We fixed an error that could arise when the [DMVCFramework](#) was installed. It could conflict on some units that were contained in Codolex framework and the DMVC framework.

The conflicting files are not included in the Codolex framework anymore.

12.6 Version 2.3.0

Highlights

AI Chat activity

Codolex provides the opportunity to make use of the latest technologies in Delphi, and with this version, ChatGTP is also on the list.

The AI Chat activity makes use of the Open AI API to ask and receive answers from ChatGTP.

All you need is an API Key and choose a model to chat with, and you can use the online tool however you like.

Edit properties - AI Chat

Properties Information

Description

Authorization key

Model

GPT 4

List of all chat messages

Content

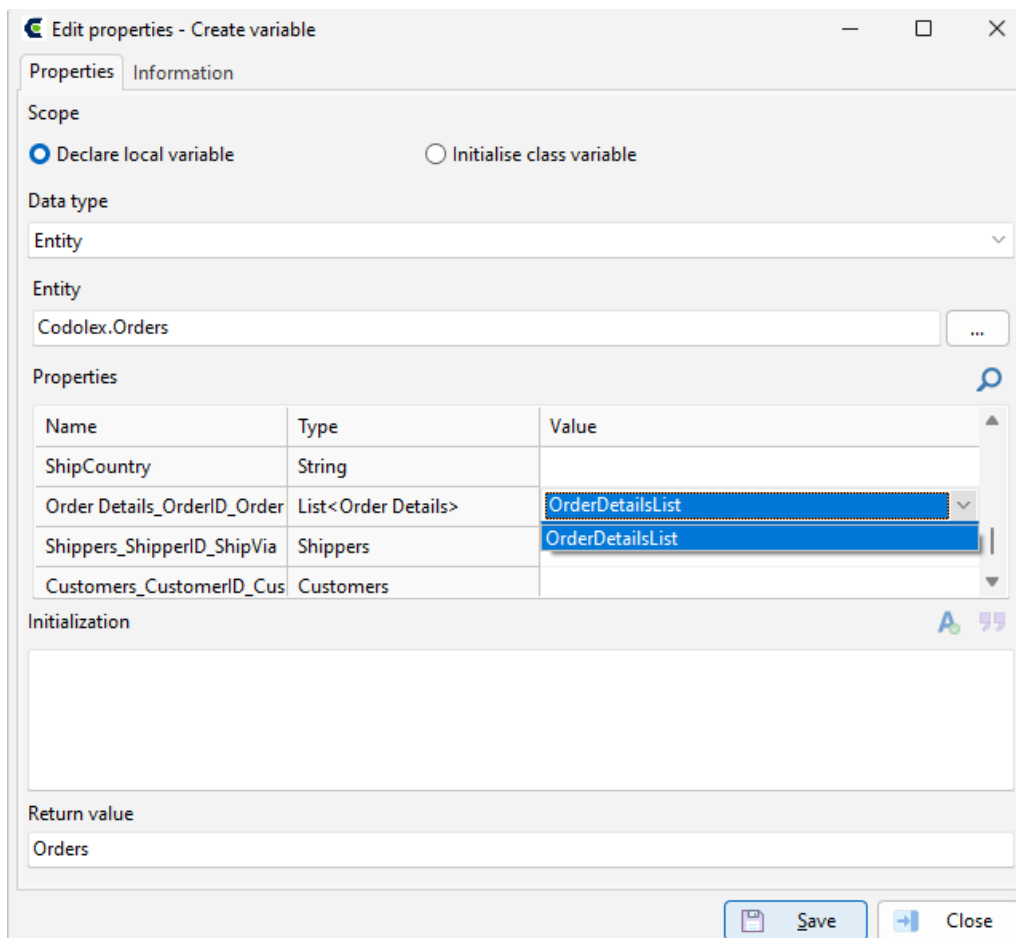
Return value

ChatMessage

Save Close

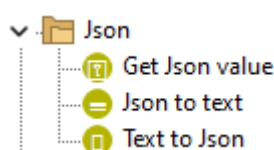
Setting associations

When building data in Codolex to save or to send, the options on setting associations were limited, or not very clear on some occasions. With Codolex 2.3 we made it possible to set associations directly with a change entity or create entity activity. This removes the hassle of setting id's manually and saving the entities before you could complete the structure.



Addition of data type JSONValue

Working with JSON got much better in Codalex 2.3 with the addition of JSON Value and some activities



These activities allows for converting JSON text into JSON object. And with the JSON object you can get values directly without converting them to entity's or searching to the text.

You can also convert a JSON back to text to save it in a file or send it with requests for example.

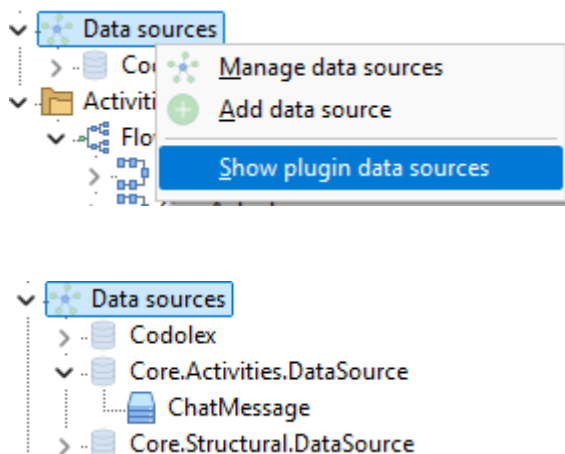
You can also use the TJSONValue as data type when creating a variable, setting the type of a parameter or defining the attributes of your entity in a non-database datasource.



Other features

Plugin datasources

Some activities from now on might return a custom entity instead of primitive values or the need to provide a custom entity. The new chat message for example, the activity want's to return more than just a string with an answer, so it will provide the chat message entity. This entity is available in Codolex by default and can be seen in the menu.



These datasources are readonly for inspection, and can be added by other activities in the future.

NexusDB Connection

If you use nexusDB as your preferred database, you can configure Ccodolex to use it. For more information on how to connect:

- [Custom database connections](#)⁴⁹
- [Nexus DB](#)⁴⁹

Improvements

1. Captions of flow activities will generate with variable names and be more clear by default.
2. Name and password fields of basic authentication in the REST activity are now parameter fields.
3. Name spaces for custom types can be directly added when declaring variables and flowclass variables.
4. Codolex Version number is now visible at RAD Studio startup
5. Stringify option is directly visible instead of hidden in a menu
6. Small improvements for the expression validator, like multiline checking and a button to validate.
7. Text search for decisions and loops

8. Filter in activity properties
9. Creation of a flow class variable is now possible for use in the flow call activity

Bugfixes

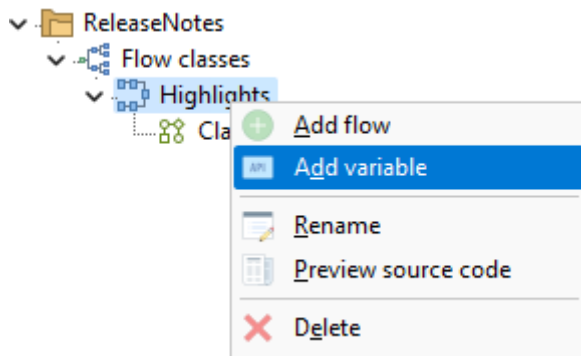
1. Solved duplicate code when using decisions and merge activities
2. Using the save to DB action on entities that have no changes will no longer result in errors
3. Using variable names that are the same as the project will no longer result in errors
4. Database params for field generators are now correctly added
5. Improved handling of dragging activities in the flow editor

12.7 Version 2.2.1

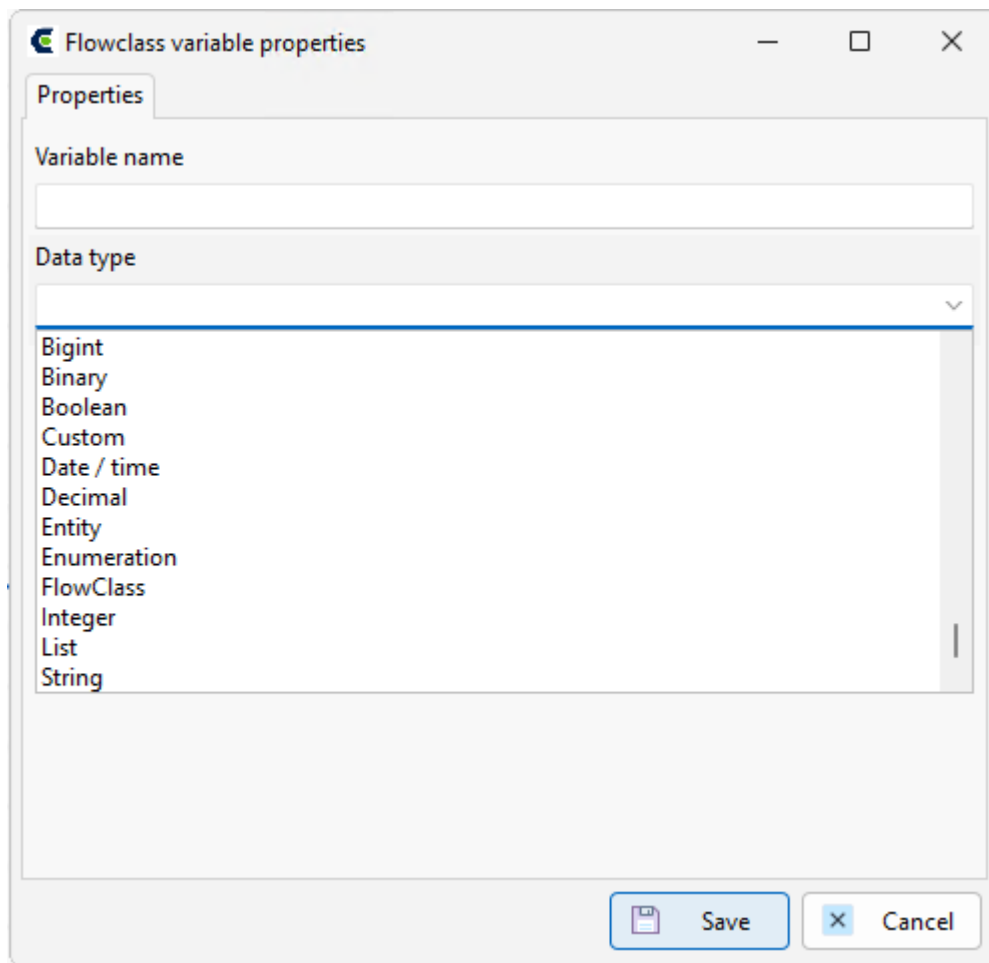
Highlights

Addition of class variable

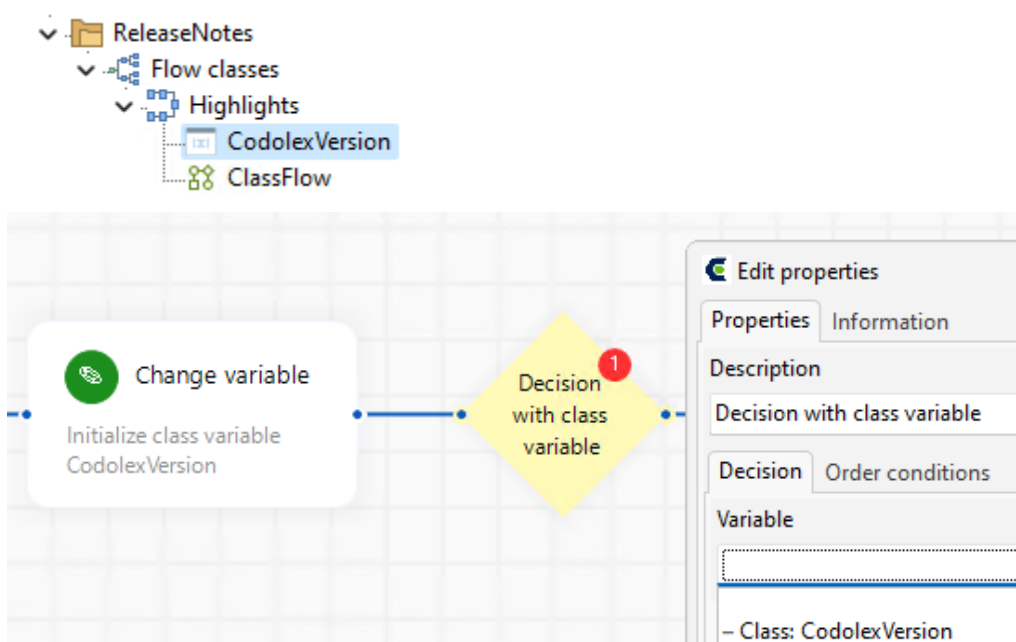
To make it easy to work with multiple flows that needs the same variable, we have added the flow class variable. this variable can be of any type, and is useable by every flow in the flow class. To add a flow variable, open the flow class menu by right clicking the flow class in the project explorer, and selecting "Add variable"



This opens the properties screen for a flow class variable, where you can set the name and type



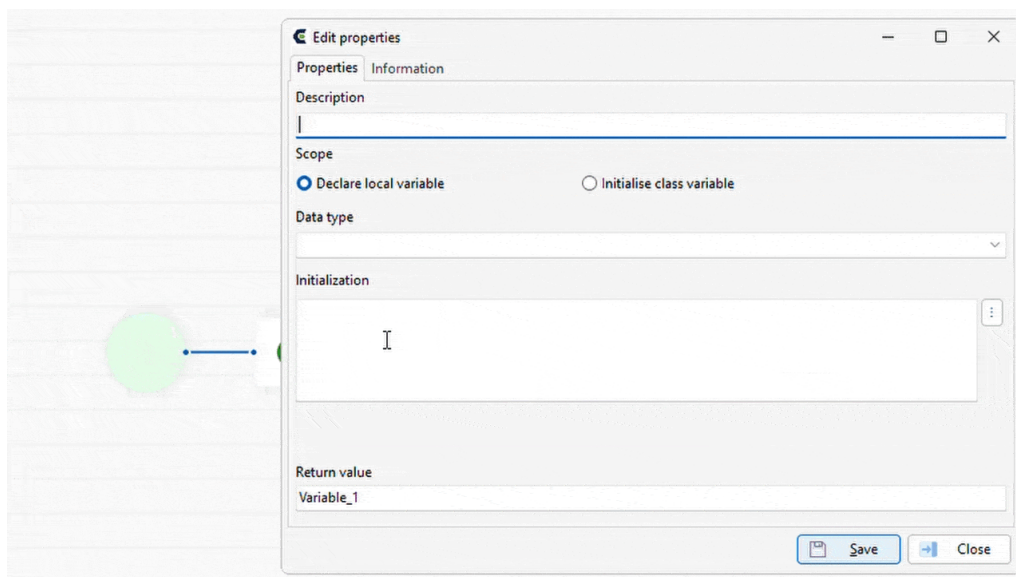
Once created, the variable needs to be initialized in a flow, and can be used in every flow, where a variable can be selected.



To delete a class variable, open the properties and select, delete variable

Expression validator

A new feature which is going to help immensely with writing expression right is the expression input evaluator. The evaluator checks if the expression results in a valid input for the property or action. For example, when setting the value of a string variable, the evaluator checks if the expression results in a string.

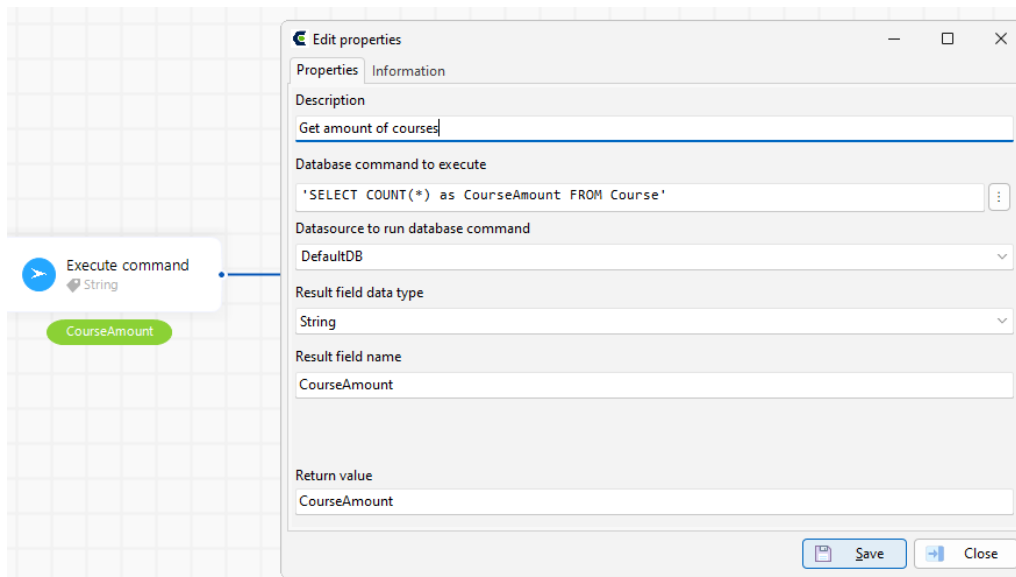


Other features

Command activity

A new activity "Execute command" has been added. This activity helps in retrieving data from the database.

The activity lets you run a command on a database and get a result directly into a variable without retrieving the result into an entity. This is great for sql commands like count or max, or other commands where only one simple result variable like a string or integer is needed.



List as attribute

For memory databases like a JSON database, it's possible to set an attribute as collection. This makes it easy to add a list to an entity, and let it export in a JSON as an array.

Name	DataType	Size	Primary key	Is collection
menuID	Integer	0	True	False
popupID	Integer	0	False	False
id	String	4	False	False
value	String	4	False	False
> Options	String	20	False	True

Improvements

Flow editor canvas improved

The canvas works better when dragging an activity to the end of the canvas, allowing for easier expansion of the canvas.

Bugfixes

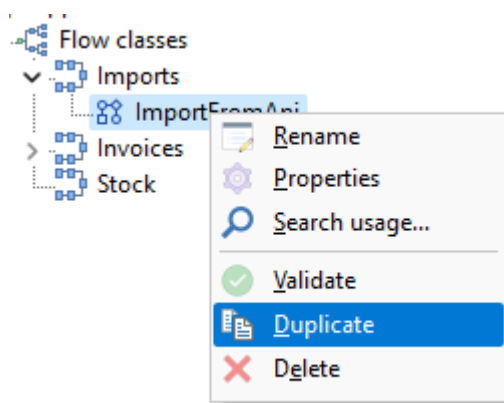
1. The canvas alignment option are available for the start and end nodes in a flow
2. Fixed an issue that made a loop not moveable in some cases
3. IsValidFileName now correctly checks the entire file name
4. Fixed an exception that could arise when dragging the create activity in a flow after searching on entity
5. Binary fields are only included in an update query when changed from now on.
6. Validation errors now move correctly with an activity when the position is changed.-

12.8 Version 2.1.0

Highlights

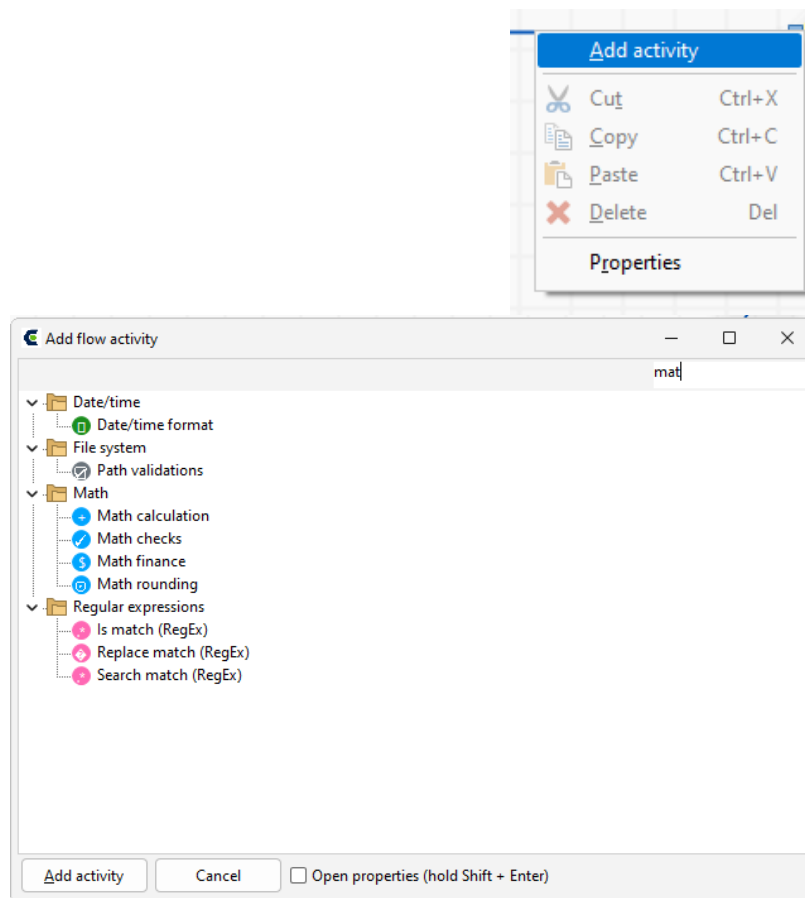
Duplicate a flow

It is now easy to duplicate an existing flow from the context menu of the flow. A dialog will appear where you can enter a new name for the flow and then a copy of the flow will be added to the flow class. This allows you to easily use the base of an existing flow for a new flow.



Add activities on a sequence

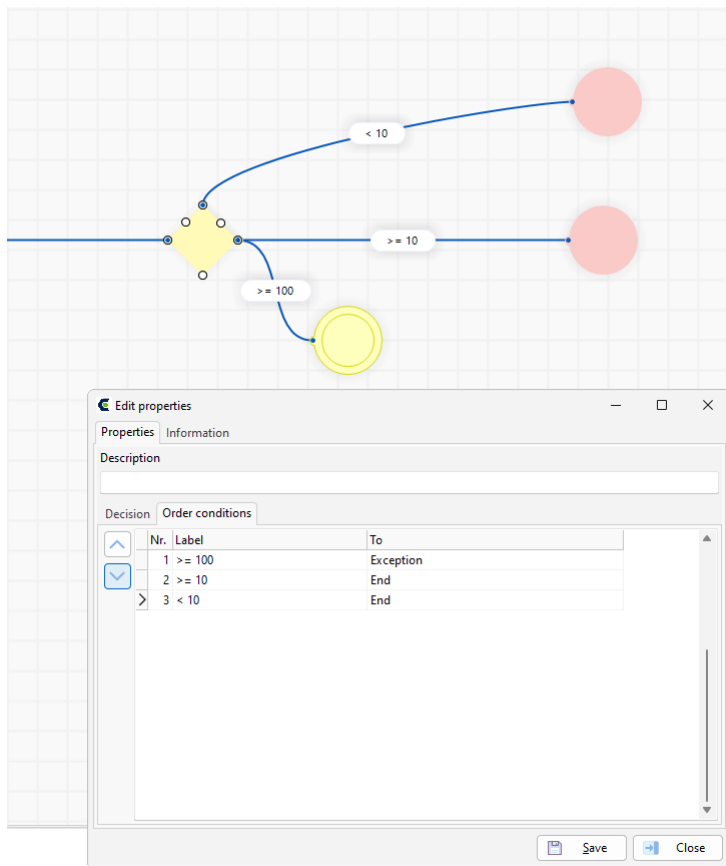
Right-click on a sequence to add an activity directly in the right place. Instead of switching between the flow and the palette, there is now a quicker option. The palette opens into a search dialog where you can type directly to find the right activity. Use the down arrow to jump to the list and select the activity. Double-click or Enter to insert the activity into the flow. Shift+Enter inserts the activity and immediately opens the Properties dialog.



Working with decisions

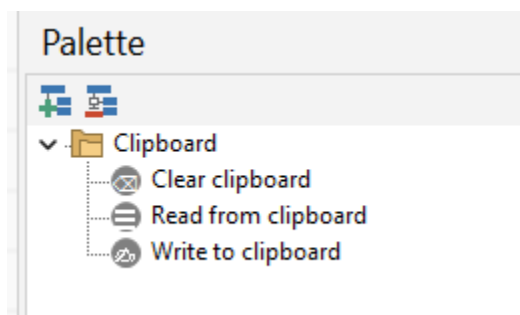
When adding a decision, the properties dialog immediately displayed the validation message that a sequence with a condition had to be added. This message no longer appears in the decision properties dialog, as you can't change anything at this point.

It is also now possible to arrange the sequences of decision conditions in a specific order. This is important for code generation. The order can be important, especially when working with numbers.



New Clipboard activities

New activities have been added to use the VCL clipboard functionality. It is now easy to write to, read from and clear the clipboard.



Improvements

1. A new Information tab has been added to the properties dialog for an activity. This tab contains an input field which allows you to change the caption/title of the activity. This helps to make the flows more readable.
2. Firebird can now be selected as the database server for a datasource. This means that a specific Firebird SQL dialect is used to generate statements for entities and database queries.
3. Generators are now imported from a database. A generator can be associated with a field in the data entity edit dialog. This generator will then always be used in insert statements for that entity. Generators are supported by the Oracle and Firebird database server options.

4. The field definitions of the Codolex dataset now include the precision and scale for numeric fields as defined in the data entity.
5. You can now also use Ctrl+D in a list of properties to apply the stringify function. For example, when creating or modifying a variable with an entity or when calling a flow with parameters.
6. It is now possible to use literals in the text of a custom query when selecting columns. For example, a fixed text or a number instead of a column name.
7. Added the ability to add and use form data parameters to REST operation parameters.
8. Validation messages produce warning messages in the generated code. Even if they are informative. It also blocks the generation of code for the activity. This has been changed. Informative validations will not block code generation and produce hint messages instead of warnings.
9. It is now possible to clear a variable selection in property dialogs.
10. Other small changes to improve the JSON import, make labels in activities clearer, fix tab order in property dialogs and make the editor more stable.

Fixes

1. Boolean fields are now converted to JSON as true and false instead of -1 and 0.
2. The select statement of a view entity containing an updatable entity is now saved correctly after changing.
3. When a field of an entity variable is used, the condition will now produce the correct code statement.
4. It is no longer possible to drag connected activities into a subflow.
5. The flow is now better displayed in the flow editor when the source code preview is open. Variable names now remain visible.
6. When adding a parameter to the REST operation, the value of the selected parameter is no longer displayed. The fields are empty.
7. Pressing Ctrl+D in an empty input no longer throws an exception.
8. If an entity field was updated and the entity instance was also used in a dataset, an error message was displayed if the modified field was not defined as a field in the dataset. This has been fixed.
9. Changes to the project, a data entity or a data flow can cause validation notifications in other data flows. These validations did not always reflect correctly in the editors of open flows.
10. The Codolex dataset now throws an exception if LoadEntities is called while the dataset is active.