

# **USER MANUAL**

© 2024 GDK Software

www.codolex.com

1. Intr	oduction 7
1.1	The idea behind codolex8
2. Get	ting started with Codolex 9
2.1	Installation10
2.2	IDE integration
2.3	Your first Codolex Application11
2.4	Codolex Concepts
2.4.1	Project explorer
2.4.2	Modules
2.4.3	Flow classes
2.4.4	Flows
2.4.5	Data sources
2.4.6	Local data sources
2.4.7	Entities
2.4.8	Search
3. Cod	le generation 23
3.1	Automatic code generation
3.2	Manual code generation24
3.3	File structure
4. Flow	ws 26
4.1	Flow handling
4.2	Start and End
4.3	Flow results
4.4	Parameters
4.5	Visibility
4.6	Code preview
4.7	Validation
E Dot	24
5. Data	a sources 34
5.1	Local data sources 35
5.2	Connecting to databases
5.3	Custom database connections
5.3.1	Nexus DB
5.4	Entities
5.5	Nullable fields
5.6	Importing data structures
5.7	IniFile Datasource

# **Table of Contents**

5.8	JSON datasource4	1
5.9	Plugin datasources 4	3
6. A	ctivities 4	4
6.1	Structural4	5
6.1.	L Parameters	5
6.1.2	2 End	6
6.1.	3 Start	.7
6.1.4	4 Sequences	8
6.1.	5 Decision	8
6.1.	5 Merge 4	.9
6.1.	7 Loop 5	0
6.1.	3 Exceptions 5	2
6.2	Core	3
6.2.	L Flow call	4
6.2.2	2 Code snippet	5
6.2.3	3 Use unit	5
6.3	Clipboard5	6
6.3.	L Write to clipboard	6
6.3.2	2 Read from clipboard 5	7
6.3.3	3 Clear clipboard 5	8
6.4	Database	9
6.4.	L Get from DB	9
6.4.	2 Save object	1
6.4.3	B Delete object	1
6.4.4	4 Transactions	2
6.	4.4.1 Start transaction6	3
6.	4.4.2 Commit transaction	4
6.	4.4.3 Rollback transaction	,4 
6.4.	Execute command	5
6.5	Date/Time	6
6.5.	Calculations	7
6.5.	2 Check	9
6.5.	Conversion	0
6.5.4	t Decode	2
6.5.	5 Encode	3
6.5.	o Format	4
6.5. C.F.		5
0.5.		0
0.5.	<ul> <li>Uuis</li> <li>Validation</li> </ul>	ð
0.5.		9
<b>b.</b> b		2
6.6.	L OpenDialog	2

0.0.2	SaveDialog	84
6.6.3	ShowDialog	85
6.6.4	Show modal form	86
6.7	Encoding	88
6.7.1	Decode	88
6.7.2	Encode	89
6.8	Entity conversion	90
6.8.1	Entity to JSON	91
6.8.2	JSON to entity	92
6.8.3	Entity to key/value	93
6.8.4	Key/value to entity	94
6.9	File system	95
6.9.1	Copy file/folder	
6.9.2	Create file/folder	97
6.9.3	Delete file/folder	
6.9.4	Exists file/folder	
6.9.5	Get path part	100
6.9.6	Get system path	103
6.9.7	Listing file/folder	104
6.9.8	Move file/folder	105
6.9.9	Path validations	106
6.9.10	Read file	111
6.9.11	Write file	112
6.10	Hashing	112
6.10.1	Hash file	113
6.10.2	Hash string/bytes	114
6.11	Import/Export	115
6.11.1	CSV export	115
6.11.2	CSV import	116
6.12	IniFile	118
6.12.1	IniFile Read	118
6.12.1 6.12.2	IniFile Read IniFile Write	118 119
6.12.1 6.12.2 <b>6.13</b>	IniFile Read IniFile Write JSON	118 119 <b>120</b>
6.12.1 6.12.2 <b>6.13</b> 6.13.1	IniFile Read IniFile Write JSON Text to JSON	118 119 <b>120</b> 121
6.12.1 6.12.2 <b>6.13</b> 6.13.1 6.13.2	IniFile Read IniFile Write JSON Text to JSON JSON to text	118 119 <b>120</b> 121 122
6.12.1 6.12.2 6.13 6.13.1 6.13.2 6.13.3	IniFile Read IniFile Write JSON Text to JSON JSON to text Get JSON value	118 119 <b>120</b> 121 122 122
6.12.1 6.12.2 6.13 6.13.1 6.13.2 6.13.3 6.14	IniFile Read IniFile Write JSON Text to JSON JSON to text Get JSON value Math	
6.12.1 6.12.2 6.13 6.13.1 6.13.2 6.13.3 6.14 6.14.1	IniFile Read IniFile Write JSON Text to JSON JSON to text Get JSON value Math Calculation	
6.12.1 6.12.2 6.13 6.13.1 6.13.2 6.13.3 6.14 6.14.1 6.14.2	IniFile Read IniFile Write JSON Text to JSON JSON to text Get JSON value Math Calculation Checks	
6.12.1 6.12.2 6.13 6.13.1 6.13.2 6.13.3 6.14 6.14.1 6.14.2 6.14.3	IniFile Read IniFile Write JSON Text to JSON JSON to text Get JSON value Math Calculation Checks Finance	
6.12.1 6.12.2 6.13 6.13.1 6.13.2 6.13.3 6.14 6.14.1 6.14.2 6.14.3 6.14.4	IniFile Read IniFile Write JSON Text to JSON JSON to text Get JSON value Math Calculation Checks Finance Rounding	118 119 120 121 122 122 122 122 124 124 129 131 136

# **Table of Contents**

6.16	Regular expressions	142
6.16.1	Options	142
6.16.2	Escape chars	142
6.16.3	Is Match	143
6.16.4	Split string	144
6.16.5	Search match	145
6.16.6	Replace match	146
6.17	String utils	147
6.17.1	String change	148
6.17.2	String check	151
6.17.3	String conversion	153
6.17.4	String find	156
6.17.5	String parts	157
6.17.6	String split	160
6.18	Variables	161
6.18.1	Create variable	162
6.18.2	Change variable	164
6.18.3	List Operations	165
6.18.4	Copy entity data	167
6.18.5	Free Object	168
6.18.6	Get by association	169
6.19	Zipping	171
6.19.1	Create/open zip file	171
6.19.2	Extract zipfile	172
6.19.3	Add to zipfile	173
6.19.4	Close zipfile	175
6.19.5	Listing zip	175
6.19.6	Read zip	176
6.20	Advanced	177
7. Crea	ating your own activity	178
71	Gotting started	170
7.1		1/ <i>5</i>
7.2		101
7.3	Tags	
7.4	CodeGen	189
7.5	Validation	192
7.6	Tests	0
8. Con	nmand line interface	194
9. Bes	t practices	196
9.1	Source control	197
9.2	Useful tips and tricks	197

# **Table of Contents**

9.3	Use multiple flows	
10. FAC	2	198
11. Rel	ease notes	201
11.1	Version 2.4.0	
11.1.1	Version 2.4.1	205
11.2	Version 2.3.0	205
11.3	Version 2.2.1	209
11.4	Version 2.1.0	213
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. <u>Click here</u> 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: <u>https://codolex.com/download</u>

### 2.1 Installation

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

Setup - Codolex 2.0.0	—		×
Select Additional Tasks Which additional tasks should be performed?		(	
Select the additional tasks you would like Setup to perform w then click Next. Additional shortcuts:	vhile installing	g Codolex,	,
Create a desktop shortcut			
	Next	Ca	ncel

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 Codolex projects and the Codolex structure. You can create a new Codolex project or open an existing one from the project explorer, which is linked to your active Delphi project. When you compile your Delphi project, Codolex 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 Codolex Application

In your first Codolex 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. 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:



You will be prompted to save the project. Create a subfolder in your Delphi project folder and give your project a name, for example CodolexAPI. Codolex will store everything in the subfolders of the folder containing the project.

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".

<b>•</b>	Codolex
File	Edit Configuration Help
	🛅 😬 <u>S</u> ave all
:=	Codolex Projects
~	🕕 <u>A</u> dd module
	Validate
	<u> G</u> enerate code
	🔆 <u>M</u> anage data sources
	<u>Remove project</u>
	Project languages

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. Codolex 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. Codolex 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.

Flow	-	-		$\times$
Properties				
Name				
Visibility				
Public	Is static			
	Save		Cance	el 🚽

The Codolex flow editor will open. We will now build the business logic to connect to the timezone api.



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 it 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 Codolex 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.

S Edit properties			- 🗆 🗙
Properties			
Description			
Message type O Warning	⊖ Error	Information	O Confirmation
Buttons			
OK Cancel	Yes No	Retry Ignore	
Message text			
JSONResult			
Return value			
DialogResult			
		ſ	Save Close

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".

Codolex	<b>4</b> ×	GetTimeForTimeZone [Flow] Project13 Unit18 •
🔓 🛅 📄 Save all	Q	🔜 🗸 📴 🗸 Search for a type
Codolex Projects Codolex Projects TimeZoneAPI TimeAPI Flow classe TimeZoneAPI TimeAPI Second	es one TimeForTim Rename Properties Search usage Validate Xalidate Delete	<pre>type type TfrmMain = class(TForm) Button1: TButton; procedure Button1Click(Sender: TObject); private {    Private declarations } public     { Public declarations } end; var ifrmMain: TfrmMain; implementation {\$R *.fmx} } procedure TfrmMain.Button1Click(Sender: TObject); begin iend; iend;</pre>
Object Inspector	<b> </b> ×	

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

15

```
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.

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

The first thing we have to do is to import the JSON structure. Luckily, Codolex 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, Codolex will open the "Import" wizard. Select JSON as the "Import" from option, give a name for the new entity that Codolex 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.

Import from data source	
mport wizard	
mport entities and relations for "TimeZone".	
mport from:	
SON ~	
ntity name:	
TimeZoneResult	

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

Codolex 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".

JSON to e	Add exception	
	Set variable TimeZoneResult as r	eturn value for flow
TimeZoneRes	Edit description	F2
	🔀 Cu <u>t</u>	Ctrl+X
	<u>С</u> ору	Ctrl+C
	Paste	Ctrl+V
	🗙 <u>D</u> elete	Ctrl+Del

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 3.

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 higest 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 Codolex 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 <u>Connecting to databases</u> 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 Codolex, entities are representations of data. This can be data from a database, but also from residual services or other sources. See Entities of more information about entities.

#### 2.4.8 Search

When you have a large Codolex 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 Codolex. 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.

En	tity usage: "PriceCalculation.CustomerPrice"				
	Document	Element	Name	Property	Text
>	🖉 CodolexDemo / CodolexDemo / PriceCalculatio	Association	CustomerPrice_Custome	FromEntity	CustomerPrice
	CodolexDemo / DemoDB / PriceCalculation / G	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 <u>command line interface</u> 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.

25

# **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.

🔶 Flow		-		×
Properties				
Name				
Collect credit card infor	mation			
Visibility				
Public	✓ Is static			
Private				
Public				
		Save	Car	ncel

### 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

Menultem				
				String
		Get name from menu item	<b></b>	-
	•	MenultemName		MenultemNa me

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 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" on an activity.

	Create variable	
Cr	Set variable Configuration	as return value for flow
- (	Cor <u>E</u> dit description	F2
	🖌 Cu <u>t</u>	Ctrl+X
	<u>с</u> ору	Ctrl+C
	Paste	Ctrl+V
	× Delete	Del
	P <u>r</u> operties	

If you set a flow result, the Codolex 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.



When calling a flow from another flow (via the Call Flow activity) or via Delphi, you need to fill in these parameters.

# 4.5 Visibility

Flow	-	$\times$
Properties		
Name		
Visibility		
Visibility Public	✓ 🗌 Is static	
Visibility Public	✓ 🗌 Is static	

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 Codolex will generate, click the preview button in the menu:

PriceCalculatio	n.GetP ice			
Close O Io	ol palette 🗻 Preview 🖉 Undo  🕥	<u>a R</u> edo		
	String			
	CustomerId			
	Integer		Check if the customer has a	
			specific price for this product	
	Producid		_	
	Get from db	Create variable	Get from db	Tri
	CustomerPrice	🔍 🗬 Boolean	List <customers></customers>	
				1
	CustomCustomer	HasCustomerPrice	CustomersList	
				$\sim$
	n TPriceCalculation GetPrice	(const ProductId: Integer: const	CustomerId: string): Double:	
1 functio		conse inoducerat integer, conse	customeria: sering): sousie,	
1 functio 2 begin				
1 functio 2 begin 3 var C	ustomCustomer: CodolexDemo.Da	taSource.PriceCalculation.ICust	omerPrice;	
1 functio 2 begin 3 var C 4 var S	ustomCustomer: CodolexDemo.Da QL :=	ataSource.PriceCalculation.ICust	omerPrice;	
1 functio 2 begin 3 var C 4 var S 5 'SE	ustomCustomer: CodolexDemo.Da QL := LECT CustomerPrice.* '+ sLine	ataSource.PriceCalculation.ICust 2Break +	omerPrice;	
1 functio 2 begin 3 var C 4 var S 5 'SE 6 'FR	ustomCustomer: CodolexDemo.Da QL := LECT CustomerPrice.* '+ sLine OM CustomerPrice AS CustomerP	ataSource.PriceCalculation.ICust 2Break + ?rice '+ sLineBreak +	omerPrice;	

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

Codolex has a comprehensive validation system, to help you find errors, warnings and hints. If you use the standalone version of Codolex, 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.

Data sources

#### 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.

## 5.1 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 "Codolex" (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 Codolex 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".

Add data source			-		>	<
Properties						
Name						
TestDatabase						
Туре						
Database					$\sim$	
Database server <ul> <li>Default</li> </ul>						
O Oracle						
O MSSQL						
Import data						
O Yes						
○ No						
		8	Save	×	Cance	el

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.

### 5.2 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.
# 5.3 Custom database connections

See this YouTube video for more information

#### 5.3.1 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 componen
                          var Connection := TDatabaseConnectionN
                          // Create a guery wrapper that uses th
                          Result := TDatabaseQueryNexusDb.Create
                        end;
 // Link the provider function to your datasource from the Codo
  // Replace MainDatabase with the name of your database datasou
 CodolexFramework.DatabaseQueryProvider['MainDatabase'] := Nexu
```

# 5.4 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.

# 5.5 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: TCodolexField <string>;</string>

# 5.6 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, rightclick on the data source and choose View Model. Codolex will open the data view including all the entities of the data source.

DataSource.PriceCalculation *						
🔁 Close 🔂 Add 🗙 Delete 🖉 Undo 🖓 Redo						
Products       Customers         OproductName       CustomerID         OproductName       CompanyName         OsupplierID       ContactName         OcategoryID       ContactTitle         QuantityPerUnit       Address         QUnitsInStock       Region         QuistonOrder       PostalCode         QeorderLevel       Country         Oiscontinued       Phone         Fax       Fax	CustomerPrice CustomerPriceld Productid CustomerId JnitPrice					

# 5.7 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 [118] IniFile Write [119]

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

🔇 Add data source	-		×
Properties			
Name			
IniFileSource			
Туре			
Memory			~
Import data			
• Yes			
⊖ No			
( <u>)</u>	Save	× c	ancel

2. Select import type 'IniFile' and use the content of the ini file as importdata S Import from data source

mport wizard	
mport entities and relations	for "IniFileSource".
mport from:	
IniFile	~
ri )	
[Keys] APIKev1=3b7714d1-4867-4e	1f-95cf-af5b9d4b3ce5
APIKey2=a3070efa-d3cf-440	09-a213-e5c6af9bf2f0
[Names]	
Name1-randomusername	
iname i = randomusemame	

# 3. Select 'Save'

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

# 5.8 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.

C Add data source		-		×
Properties				
Name				
JSONMenuDataSource				
Туре				
Memory				~
Import data				
O Yes				
⊖ No				
(	8	Save	×	Cancel

1. Create new datasource



#### Import wizard

Import entities and relations for "JSONMenuDataSourc

 $\sim$ 

Import from:

JSON

Entity name:

RestaurantMenu

{"menu": {	
"id": "file",	
"value": "Fi	le",
"popup": {	
"menuiter	m": [
{"value":	"New", "onclick": "CreateNewDoc()"},
{"value":	"Open", "onclick": "OpenDoc()"},
{"value":	"Close", "onclick": "CloseDoc()"}
1	
}	
33	

#### 2. Import json structure.

RestaurantMenulD menulD	menu menulD
	value popupID
menuitem menuitemID value onclick	popup popupID
popupID	

3. Resulting data structure

These entities are now Codolex entities available for use in flows. More information on how to work with JSON: Json 120 Entity conversion 50

# 5.9 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

# 6 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 with you want to develop your own activity.

# 6.1 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 45
- <u>End</u> 46
- <u>Start</u> 47
- <u>Sequences</u> 48
- <u>Decision</u> 48
- <u>Merge</u> 49
- <u>Loop</u> 50
- Exceptions 52

## 6.1.1 Parameters

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



Parameter for flow

When calling a flow from another flow (via the  $\underline{\text{Flow call}}_{54}$  activity) or via Delphi. you need to fill in these parameters.

F	low					
Parameters						
	Parameters					
	Name	Туре	Value			
	Code	String	п			

Call flow action

# 6.1.2 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).

CurrencyConfig
Configuration

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.

C Edit properties	-	$\times$
Properties Information		
Description		
✓ Flow is function		
Variable		
		$\sim$
Configuration		
Configuration2		

# 6.1.3 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



#### 6.1.4 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.



#### 6.1.5 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.Strin
gParts();
    end
    else
```

enu			
Resulting code			

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

	Fals	f Call flow Call flow	
		€ Edit properties - □ ×	
	$\sim$	Description	
-•	Has ini file?	Has ini file? Variable	
		HaslniFile	
		Save Close	

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.

# 6.1.6 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.



# 6.1.7 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	O For	O Forin			
Loop variable name					
Variable with collection					
					~
		<u> </u>	ave	→	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.

Create variable	Get from db	
Create variable	Get from db	
🖤 🧬 Boolean	Get indiff db	
IsLoaded	CustomerViewList	
	IsLoaded	IsLoaded CustomerViewList

Loop in flow

```
begin
  for var i := 0 to Max do
  begin
   var IsLoaded: Boolean;
   IsLoaded := True;
    var CustomerViewList:
ICodolexList<Project.DataSource.Codolex.ICustomerView>;
   var SQL :=
      'SELECT CustomerView.* '+ sLineBreak +
      'FROM CustomerView AS CustomerView ';
    var Params: IDatabaseParams := TDatabaseParams.Create;
    CustomerViewList :=
Project.DataSource.Codolex.CustomerViewDataBroker.GetList(SQL,
Params);
  end;
end;
```



# **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





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



#### 6.1.8 Exceptions

You can use Exceptions to catch possible errors in the application. You can add exception handling to activities in Codolex. 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.Sav
e(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;
```

Resulting code

# 6.2 Core

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



- <u>Flow call</u> 54
- <u>Code snippet</u> 55
- <u>Use unit</u> 55

# 6.2.1 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.

🔇 Edit properties - Cal	ll flow		_	×
Properties Information	ו			
Flow				
Parameters				
Class instance to use (op	ptional)			
CoreActivities				~
Parameters				Q
Name	Туре	Value		
Code	String			
Dates	List <datetime></datetime>	DatesList		
Shippers	Shippers	Shippers		
			Save	Close



```
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 an be defined.

© 2024 GDK Software

Return value			
ParametersResultList			
		<u>S</u> ave	Close

*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...* 

# 6.2.2 Code snippet

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

The code snippet parses code directly into the generated flow.

• Create variable	•• (I) Code snippet •• (I)	Math rounding		
DecimalValue		RoundingValue		~
	C Edit properties		- 0	×
	Properties Information			
	Code Snippet			
<pre>1 procedure TCoreActivities.Snippet; 2 begin 3 var DecimalValue: Double; 4 DecimalValue := 1.2; 5 //Code snippet start 6 DecimalValue := DecimalValue * 2; 7 //Code snippet end 8 var RoundingValue: Integer; 9 RoundingValue := Ceil(DecimalValue); 10 end;</pre>	//Code snippet start DecimalValue := DecimalValue * 2; //Code snippet end			
			Save 3	lose

# 6.2.3 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.

Use unit sys	unit	Get working directory	•	
	E Edit prop Properties Description Use unit sys	erties Information	-	
<pre>1 procedure TCoreActivities.FlowCall; 2 begin 3 var WorkingDir: string; 4 WorkingDir := getCurrentDir; 5 end;</pre>	Namespace System.Syst Uses sectio Interfac	Utils on :e	Implementation	

*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

# 6.3 Clipboard

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



- <u>Clear clipboard</u> 58
- <u>Read from clipboard</u> 57
- Write to clipboard 56

# 6.3.1 Write to clipboard

Write a value to the users clipboard with this activity Uses <u>Vcl.Clipbrd.TClipboard</u>

C Edit properties	-	$\times$
Properties Information		
Description		
Clipboard value		
		$\sim$
BinaryImage		
StringVariable		

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

#### 6.3.2 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.



begin
var ClipBoardImage: ICodolexBinary;
var ClipBoard := TClipBoard.Create;
try
ClipBoard.Open;

```
58
```

```
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;
```

#### 6.3.3 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* <u>Vcl.Clipbrd.TClipboard</u>

The activity has no properties to fill.





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

Resulting code

# 6.4 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.

- Database
   Delete object
   Execute command
   Get from db
   Save object
   Transaction commit
   Transaction rollback
   Transaction start
- Get from DB 59
- <u>Save object</u> 61
- Delete object 61
- Transactions 62
- Execute command 65

# 6.4.1 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.

0	Edit pr	operties - Get from	ı db				—	$\times$
Pro	perties	Information						
×	ENTIT	( - [Customers]						
Co	dolex.C	ustomers						
×	SELEC	Г						
	SELEC FROM	T Customers.* Customers						
×	WHER	E						
≽	ORDEF	RBY						
×	LIMIT	RESULTS						
c	No	⊖ First	◯ Limited	Offset	0	Record cou	nt 0	
Retu	urn valu	e						
Cu	stomers	List						
							Save	Close

Activity properties

```
begin
  var CustomersList:
ICodolexList<HelpAndManualScreenshots.DataSource.Codolex.ICustomers>;
  var SQL :=
    'SELECT Customers.* '+ sLineBreak +
    'FROM Customers AS Customers ';
  var Params: IDatabaseParams := TDatabaseParams.Create;
    CustomersList :=
HelpAndManualScreenshots.DataSource.Codolex.CustomersDataBroker.GetLis
t(SQL, Params);
end;
```

#### Resulting code

To get started, select an entity from the search list. Codolex 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.

# 6.4.2 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.

🧲 Edit proj	perties		- (	-	<
Properties	Information				
Description					
Variable					
					4
Shippers					
ShippersLi	it				
				-	
		<u>S</u> ave		Close	

Activity properties

```
begin
  HelpAndManualScreenshots.DataSource.Codolex.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.

# 6.4.3 Delete object

Delete object that is present in a database.

Edit properties	_	$\times$
Properties Information		
lescription		
/ariable		
Shippers		 $\sim$
	ave	Close
		ciose

#### Activity properties

```
begin
HelpAndManualScreenshots.DataSource.Codolex.ShippersDataBroker.Delet
e(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.

#### 6.4.4 Transactions

Transactions provide a secure way of working with databases. There are three possible activities: start, commit and rollback. Use these together with **Exceptions** set to ensure data is saved correctly.



- Start transaction 63
- <u>Commit transaction</u>
- End transaction 64

# 6.4.4.1 Start transaction

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

C Edit properties	_		$\times$			
Properties Information						
Description						
Select a datasource to start a transaction						
			$\sim$			
Codolex						

## Activity properties





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

It is possible to start multiple transactions on one database.

## 6.4.4.2 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.

#### 6.4.4.3 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.

# 6.4.5 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.

		C Edit properties	-		$\times$
		Properties Information			
		Description			
		Count shippers			
		Variable			
		ShippersList			$\sim$
		List operation			
	List operation	Count			~
-• Get from db @ List <shippers></shippers>	🥏 🧬 Integer 🔹 🔹				
c	ount shippers				
ShippersList	ShippersCount				
	Shippersecond				
		Return value			
		ShippersCount			
		2 🖤	ave	→ Cl	lose

In the case you only want to know the count, and not perform any other action with the records, this sollution 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.

# Activities

S Edit properties - Execute command	_	×
Properties Information		
Database command to execute		a II
'SELECT COUNT(*) as shipperscount FROM shippers'		
Datasource to run database command		
Codolex		~
Result field data type		
Integer		$\sim$
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.

# 6.5 Date/Time

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

Date/time
 Date/time calculation
 Date/time check
 Date/time conversion
 Date/time decode
 Date/time encode
 Date/time format
 Date/time from
 Date/time operation
 Date/time utils

Calculations

Check [69] Conversion [70] Decode [72] Encode [73] Format [74] From [75] Operation [76] Utils [78] Validation [79]

# 6.5.1 Calculations

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

Uses <u>System.DateUtils</u>

S Edit properties		-	×
Properties Information			
Description			
Calculation type			
Date between			~
Date between			
Date span			
Same date time			
Variable date from	Variable date to		
DateTimeFrom ~	DateTimeTo	$\sim$	

## 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 witch 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

#### 6.5.2 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 <u>System.DateUtils</u>* 

Datetime value	
DateToCheck	~
Check to preform	
	~
AM	
PM	
InLeapYear	
SameDay	
Today	

#### <u>IsAM</u>

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;
```



#### <u>IsPM</u>

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

## 6.5.3 Conversion

This activity converts a datetime variable into a string or integer variable, depending on the method. *Uses <u>System.DateUtils</u>* 

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.

## <u>Unix</u>

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
```

#### <u>ISO8601</u>

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 := DateToIS08601(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

#### 6.5.4 Decode

The decode datetime activity provides the possibility to get specific part(s) of a date into integer(s).
### Uses <u>System.DateUtils.DecodeDateTime</u>

S Edit properties - Date/t	Edit properties - Date/time decode									
Properties Information										
Input variable date time										
Date						~				
Output variable year	Output variable month	Output variable day								
YearVar $\sim$	×	DayVar	~							
Output variable hour	Output variable minute	Output variable second								
~	~		~							
				<u> </u>		Class				
				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

#### 6.5.5 Encode

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

Uses <u>System.DateUtils.EncodeDateTime</u>

## Activities



Activity properties

```
begin
  var Date: TDateTime;
  var Year := 2024;
  var Month := 01;
  var Day := 01;
  var Hour := 00;
  var Minute := 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.

### 6.5.6 Format

Format the date time into a string. The format can be customized with a string and settings. Uses <u>System.SysUtils.FormatDateTime</u>

# Activities

S Edit properties - Date/time format		-		$\times$
Properties Information				
Format				
уууу-MM-dd hh:mm:ss				
Date value				
DateValue				~
FormatSettings value				
				~
Return value				
Variable				
	B	Save	→	Close
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: <u>Embarcadero wiki FormatDateTime</u>

### **FormatSettings value**

An optional input to provide datetime format settings for this format. Variable type: <u>TFormatSettings</u>

### 6.5.7 From

The date time from activity converts a string to a date time with the current local date/time format. Uses <u>System.SysUtils.StrToDateTime</u>

lit properties - Date/time from	-		×
Properties Information			
Datetime value			
FormatedDate			~
Format settings			
FormatSettings			~
Return value			
Variable			
		_	-
💾 Sa	ve	-	Close

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: <u>TFormatSettings</u>

### 6.5.8 Operation

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

Uses <u>System.DateUtils</u> Uses <u>System.SysUtils</u>

S Edit pr	operties - Date/time operation		_	×
Properties	Information			
Datetime v	alue			
Date				~
Operation	уре			
Increase				~
Date time f	rame			
Week				~
Variable inc	rease number			
DateChan	gelnteger			~
Return valu	e			
Increased	Date			 
	(	B	Save	Close

#### 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

### 6.5.9 Utils

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

Uses <u>System.DateUtils</u> Uses <u>System.SysUtils</u>

🔇 Edit p	roperties - Date/time utils			-		×
Properties	Information					
Date/time	function to use					
Date						~
Return val	ue					
variable						
				Save	•	Close
Activity	properties					
begin var Var:	Variable: TDateT iable := Date;	ime;				
CIIU.						

Resulting code

### **Date**

Gets the current date without the time of the day.

#### <u>Now</u>

Gets the current date time to the millisecond.

### <u>Time</u>

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.

### 6.5.10 Validation

This activity can be used to validate if a date is valid when given integer values for its parts. Uses <u>System.DateUtils</u>

S Edit prope	erties					—	×
Properties In	formation						
Description							
Validation type	e						
Date time							~
Year			Month		Day		
		÷		÷			:
Hour			Minute		Second		
		÷		÷			:
Return value							
IsValid							
						571/0	Close
						Jave	Close

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;

<u>Date time</u>

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);
```



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;

## 6.6 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.



<u>OpenDialog</u> ଛ <u>SaveDialog</u> ଊ <u>ShowDialog</u> ଊ <u>Show modal form</u> เ⊛ิ

### 6.6.1 OpenDialog

The open dialog file can be used to let the user select file(s) from the explorer.

uses <u>Vcl.Dialogs.TFileOpenDialog</u>

S Edit pr	operties			_	×
Properties	Information				
Description					
1					
Default file	path				
					:
Options					
	files				
🗌 Path mu	ist exist				
File mus	t exist				
Pick fold	lers				
Store select	ed file(s) in				
ReturnList					~
Return valu	ie				
IsExecuted	l				
				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;
```

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.

### 6.6.2 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 <u>Vcl.Dialogs.TFileSaveDialog</u>

S Edit properties	_		×
Properties Information			
Description			
<u> </u>			
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 := 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.

#### 6.6.3 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



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



The message text needs to be a string.

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

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.

### 6.6.4 Show modal form

Show modal form can be used to open any TCustomForm available in the project.

uses <u>Vcl.Forms.TCustomForm.ShowModal</u>

S Edit pro	operties	-	$\times$
Properties	Information		
Description			
Namespace			
subfolder.	clformclass		
Form class			
TFormClas	sName		
Initializatio	n code		
//code t	o initialize form		:
Return valu	e		
ModalResu	lt		
		Save	Close

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.

## 6.7 Encoding

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

C Edit properties			-	×
Properties				
Description				
<u> </u>				
Encoding type				
O Base64	<b>O</b> HTML	OURL		
Variable to BaseEncoding				
				$\sim$
Return value				
Variable				
			21/2	Close

#### 6.7.1 Decode

Decode a string with a decoding protocol uses <u>System.NetEncoding.TNetEncoding.Decode</u>

S Edit pro	operties			-		×
Properties	Information					
Description						
Encoding ty	pe					
URL						$\sim$
Value to en	ode					
StringToDe	code					~
Return valu	e					
DecodedSt	ring					
					_	
			[ 🖺	Save	-	Close

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:

- <u>URL</u>
- <u>HTML</u>
- <u>Base64</u>

```
6.7.2 Encode
```

Encode a string with an encoding protocol uses <u>System.NetEncoding.TNetEncoding.Encode</u>

🔇 Edit pro	operties			-	×
Properties	Information				
Description	l.				
Encoding t	vpe				
Base64					$\sim$
Value to en	code				
StringToEn	code				$\sim$
Return valu	ie				
EncodedSt	ring				
	-				
				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:

- <u>URL</u>

- <u>HTML</u>

- Base64

## 6.8 Entity conversion

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



Entity to JSON [91] JSON to entity [92]

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

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.

### 6.8.1 Entity to JSON

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

uses <u>System.JSON.TJSONObject</u>

S Edit properties	- Entity to JSON		-		×
Properties Inform	nation				
Variable					
Task					~
Return type					
<ul> <li>String</li> </ul>	◯ JSON object				
Return value					
JSON					
			Save	•	Close
Activity prop	perties				

```
begin
var JSON: string;
var Adapter:
ICodolexEntityJSONAdapter<HelpAndManualScreenshots.DataSource.CodolexD
atasource.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.

### 6.8.2 JSON to entity

The JSON to entity activity can be used to parse a JSON string into an entity. *uses <u>System.JSON.TJSONObject</u>* 

Stati properties - JSO	N to entity		_		×
Properties Information					
Expected results					
<ul> <li>Single</li> </ul>	OMultiple				
Entity					
CodolexDatasource.Task	4				
Variable					
StringWithJSON					~
Return value					
TaskFromJSON					
			Save	₽	Close
Activity propertie	s				

© 2024 GDK Software

```
begin
  var TaskFromJSON:
HelpAndManualScreenshots.DataSource.CodolexDatasource.ITask;
 TaskFromJSON := nil;
 var ResultCollection :=
TCodolexCollections.CreateList<HelpAndManualScreenshots.DataSource.Cod
olexDatasource.ITask>;
  var ParsedJson := TJsonObject.ParseJSONValue(StringWithJSON, True,
True);
  try
    var Entity:
HelpAndManualScreenshots.DataSource.CodolexDatasource.ITask;
    var Adapter:
ICodolexEntityJSONAdapter<HelpAndManualScreenshots.DataSource.CodolexD
atasource.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.

### 6.8.3 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

S Edit properties - Entity to	o key/value			—		$\times$
Properties Information						
Entity variable						
Task						~
Return type						
◯ String		TStringList				
Return value						
TaskKeyValues						
			💾 Sa	ve	-	Close
Activity properties						

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

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

### 6.8.4 Key/value to entity

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

uses <u>System.Classes.TStringList</u>

resulting code

🔇 Edit pr	operties - Key/value to entity		_		×
Properties	Information				
Variable to	convert				
KeyValueT	ask				~
Entity					
CodolexDa	atasource.Task				
Return valu	ie				
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.

### 6.9 File system

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

File system
 Create file / folder
 Copy file / folder
 Delete file / folder
 Exists file / folder
 Listing file / folder
 Move file / folder

- Read file
  Write file

Copy file/folder Create file/folder Delete file/folder Exists file/folder Get path part Get system path Listing file/folder Move file/folder Path validations Read file Mrite file 12

### 6.9.1 Copy file/folder

The copy/folder file activity helps with duplicating resources in the system folders from one place to another. *uses* <u>System.IOUtils.TDirectory.Copy</u> *uses* <u>System.IOUtils.TFile.Copy</u>

Properties Information			
Part			
• Folder O File			
Source path			D
'C:\\Codolex\TestData'			
Destination path			
'C:\\Codolex\TestDataCopy'			
Options			
Do when folder exists			
	Save	📑 Clo	se
Activity properties			
<pre>begin if TDirectory.Exists('C:\\Codolex\TestData') the TDirectory.Move('C:\\Codolex\TestData', 'C:\</pre>	1		
<pre>\Codolex\TestDataMoved'); end;</pre>			
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 exits.

#### 6.9.2 Create file/folder

The create file/folder activity can be used to create files or folder on the local system uses <u>System.IOUtils.TDirectory.CreateDirectory</u> uses <u>System.IOUtils.TFile.AppendAllText</u>

# Activities



Activity properties

```
begin
  TDirectory.Copy('C:\...\Codolex\TestData', 'C:\...
\Codolex\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 exits.

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.

### 6.9.3 Delete file/folder

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

Uses <u>System.IOUtils.TDirectory.Delete</u> Uses <u>System.IOUtils.TFile.Delete</u>

Properties	Information					
Part						
O Folder	⊖ File					
Path					0	
'C:\	\Codolex\TestData'					
Options						
🗌 Do wh	en folder exists 🗌 Recurs	ive				
				Save	→	Close
Activity	Properties					

```
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 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.

### 6.9.4 Exists file/folder

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

uses <u>Sys</u>	<u>stem.</u>	<u>IOUtils</u>	<u>.TDirec</u>	<u>ctory.</u>	<u>Exists</u>							
uses <u>Sys</u>	<u>stem.</u>	<u>IOUtils</u>	TFile.E	<u>-xists</u>								
~												
S Edit pr	ropertie	es - Exists f	ile / folde	er					-			×
Properties	Infor	mation										
Part												
Folder	r C	) File										
Path										Q	A	IJ
'C:\	. \Codo	lex\Test	Data'									
Patura val	lue											
FolderEvis	ite											
TOIGCIEXIS	303											_
								Sav	P		Clo	se
Activity	nror	ortios						244	-		0.0	
Activity	ριομ	entes										
hegin												

TDirectory.Delete('C:\...\Codolex\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.

### 6.9.5 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* <u>System.IOUtils.TPath Methods</u>

Properties Information		
Path part to return		
File name		~
Path to use		
FilePath		~
Return value		
FileNameOfPath	 	 
	Save	Close
A ativity a super a still a		

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.GetExtenstion(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.

### 6.9.6 Get system path

Get system path helps with getting the path for default folders on the system uses <u>System.IOUtils.TPath Methods</u>

Properties Information				
Path to return				
Home path				~
Return value				
HomePath				
	_		_	
		Save	-	Close

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

### 6.9.7 Listing file/folder

The listing file/folder activity list all the files or folder in a directory uses <u>System.IOUtils.TDirectory.GetDirectories</u> uses <u>System.IOUtils.TDirectory.GetFiles</u>

Properties Information		
Part to list		
◯ Folder O File		
Directory		<b>A</b> 🗓
'C:\\Codolex\Documentation'		
Return value		
FileList		
	💾 Save	- Close
Activity properties		

begin
<pre>var FileList: ICodolexList<string>;</string></pre>
<pre>FileList := TCodolexCollections.CreateList<string>;</string></pre>
<pre>FileList.AddRange(TDirectory.GetFiles('C:\</pre>
<pre>\Codolex\Documentation'));</pre>
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 <u>EDirectoryNotFoundException</u> is thrown

105

### 6.9.8 Move file/folder

The move file/folder activity moves files and folders from one place to another on the local file system Uses <u>System.IOUtils.TDirectory.Move</u> Uses <u>System.IOUtils.TFile.Move</u>

Properties Information		
Part		
• Folder File		
Source path		
'C:\\Codolex\TestData'		
Destination path		<b>A</b> 💷
'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 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



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 exits.

### 6.9.9 Path validations

Path validations can be used to check on different parts if a path valid uses <u>System.IOUtils.TPath Methods</u>

S Edit properties - Path validations	_	$\times$
Properties Information		
Validation to execute		
ls valid path		~
Path or string to validate		
PathToValidate		~
Use wildcards		
Return value		
Variable		
	Save	Close

#### Activity properties

```
begin
var Variable: Boolean;
try
var FileNamePart := TPath.GetFileName(PathToValidate);
var PathPart := TPath.GetDirectoryName(PathToValidate);
var IsValidFileName := TPath.HasValidFileNameChars(FileNamePart,
False);
var IsValidPathPart := TPath.HasValidPathChars(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\Codolex\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 then the max path length of 260 chars.

Example:

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

### 6.9.10 Read file

The read file activity can be used to read content from a file uses <u>System.IOUtils.TFile.ReadAllText</u> uses System.Classes.TStrings.LoadFromFile

🔇 Edit properties - Read file		_		×
Properties Information				
File				
FileName				~
Data type				
String				$\sim$
Return value				
FileContent				
	-			
	B	Save	→	Close
Activity properties				

```
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.

### 6.9.11 Write file

The write file activity can be used to write or append data to a file uses <u>System.IOUtils.TFile.WriteAllText</u> and <u>System.IOUtils.TFile.AppendAllText</u> uses <u>System.Classes.TStrings.SaveToFile</u>

S Edit properties - Write file		_	$\times$
Properties Information			
Variable with content			
BinaryValue			~
File			
FileName			~
	_		
		Save	Close
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.

## 6.10 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. Codolex supports MD5, SHA1 and SHA2 hashing algoritms.



Hash file [113] Hash string/bytes [114]

### 6.10.1 Hash file

The hash file activity hashes the contents of a file and returns it as string or bytes array variable *uses* <u>System.Hash</u>

🔇 Edit pr	operties - Hash file		_	×
Properties	Information			
Hash type				
MD5				~
Variable file	path			
FileName				~
Returns O String	⊖ Bytes			
Return valu	e			
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\_521
- SHA2\_521\_224
- SHA2\_512\_256

### 6.10.2 Hash string/bytes

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

uses <u>System.Hash</u>

S Edit pro	operties - Hash string/bytes		-	×
Properties	Information			
Hash type				
MD5				~
Variable				
Content				~
Returns String	● Bytes			
Return valu	e			
Variable				
		2)	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

# 6.11 Import/Export

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



CSV export 115 CSV import 116

### 6.11.1 CSV export

The CSV export activity can be used to export a list of entities to a CSV file. Uses <u>System.Classes.TStringList</u> Uses <u>System.Classes.TStrings.SaveToFile</u>

S Edit properties - CSV export	-	×
Properties Information		
Entities to export		
ProductsList		~
File location		
Variable		~
Separator		
<u>,                                     </u>	 	 
Exclude fields		
	Save	Close
Activity properties	 	 



```
var CSVList := TStringList.Create;
 try
    CSVList.Add('ProductID, ProductName, SupplierID, CategoryID, QuantityP
erUnit,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

### 6.11.2 CSV import

The CSV Import activity can be used to import entities directly from a csv file Uses <u>System.Classes.TStringList</u> Uses <u>System.IOUtils.TFile.ReadAllLines</u>

Scheit Brop	erties - CSV import	_	×
Properties I	nformation		
Entity to impo	ort		
Codolex.Proc	lucts		
Path to impo	tfile		
ImportFileNa	ime		~
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>.Cr
eate;
    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.Codolex.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;</pre>
```

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.

# 6.12 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 Read

### 6.12.1 IniFile Read

The IniFile Read activity can be used to get data from a local ini file into a variable. uses <u>System.IniFiles.TIniFile</u>

S Edit properties - Ini	file read		_		×
Properties Informatio	n				
Variable for INI values					
keys					~
Path to INI file					
IniFilePath					~
Default values					Q
Name	Туре	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;
```



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

### 6.12.2 IniFile Write

The IniFile Write activity can be used to store data in a local ini file from a variable

uses <u>System.IniFiles.TIniFile</u>

S Edit pr	operties - Ini file write		-	×
Properties	Information			
Path to IN	file			
IniFilePat	1			~
Variable fo	r INI values			
keys				~
			Save	Close

Activity properties

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



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

# 6.13 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.



TextToJSONJSON to textGet JSON value

For more information about JSON JSON datasource Entity conversion

### 6.13.1 Text to JSON

The text to JSON Activity parses a string into a TJSONValue object. *uses <u>System.JSON.TJSONObject.ParseJSONValue</u>* 

C Edit properties - Text to Json	_		$\times$
Properties Information			
<pre>Properties Information JSON text '{"menu": {' + ' "id": "file",' + ' "value": "File",' + ' "popup": {' + ' "menuitem": [' + ' {"value": "New", "onclick": "CreateNewDoc()"},' + ' {"value": "Open", "onclick": "OpenDoc()"},' + ' {"value": "Close", "onclick": "CloseDoc()"}' + ' ]' + ' }' '}'</pre>		A	77
Return value			
JSON			
	<u>S</u> ave	Clo	se

#### Activity properties



Resulting code

### 6.13.2 JSON to text

Use the JSON to Text activity to format a JSONValue to a string uses <u>System.JSON.TJSONValue</u>

S Edit pr	operties - Json to text		-		×
Properties	Information				
Variable wi	th JSON object				
JSON					~
Return valu	Je				
JSONText					
			Save	€	Close
Activity	properties		 		
<pre>begin var JSON end;</pre>	JSONText: string; IText := JSON.Format;				

Resulting Code

### 6.13.3 Get JSON value

The get JSON Value activity can be used to get a variable from an existing JSON Value.

uses <u>System.JSON.TJSONValue</u>

S Edit properties - Get Json value	_		×
Properties Information			
Variable with JSON object			
JSON			$\sim$
Key name with value		A	
'menu'			
Data type			
JSONValue			$\sim$
Return value			
MenuJSON			
	Save	Clo	se
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;
<pre>JSON := TJsonObject.ParseJsonValue('{"menu": {' +</pre>
' "id": "other",' +
' "value": "File",' +
' "popup": {' +
' "menuitem": [' +
<pre>' {"value": "New", "onclick": "CreateNewDoc()"},' +</pre>
<pre>' {"value": "Open", "onclick": "OpenDoc()"},' +</pre>
<pre>' {"value": "Close", "onclick": "CloseDoc()"}' +</pre>
' ] ]' +
'_}' +
'}}');
var menu:
HelpAndManualScreenshots.DataSource.JSONMenuDataSource.Imenu;

```
var JsonAdapter:
ICodolexEntityJSONAdapter<HelpAndManualScreenshots.DataSource.JSONMenu
DataSource.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;
```

# 6.14 Math

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



Calculation 124 Checks 129 Finance 131 Rounding 136

### 6.14.1 Calculation

The calculation activity contains a set of math functions. Uses <u>System</u> Uses <u>System.Math</u>

# Activities

🔇 Edit pi	operties - Math calculation			_		×
Properties	Information					
Calculatio	n type					
Absolute	/alue					~
Number v	alue					
varDecim	al					$\sim$
Return val	Je					
Calculatio	n					
		 	_			
				<u>S</u> ave	→	Close

Activity properties

possible calculations:

### **Absolute value**

Needs an interger 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

### Square root

Needs an integer and returns the squared value.

```
Example:
DecimalValue = 7.5
begin
var ResultValue: Double;
ResultValue := Sqr(DecimalValue);
```

ResultValue = 56.25

end;

### Square root double

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 logariths 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

### 6.14.2 Checks

The Checks activity returns a boolean value for a validation check of a number. *uses* <u>System.Math</u>

uses <u>system.math</u>

S Edit properties - Math checks		_	×
Properties Information			
Type of check			
Is infinite			~
Number to check			
IntegerValue			~
Return value			
CheckValue			
( F	2	Save	Close
Activity properties			

Possible checks:

#### <u>Is Infinite</u>

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

### <u>ls zero</u>

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.

### 6.14.3 Finance

The finance activity includes the financial activities delphi has to offer. *uses <u>Business And Finance Routines</u>* 

S 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);
```

# Activities

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]; Rate =0.20

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

Result = 35.49...

### 6.14.4 Rounding

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

S Edit properties - Math rounding	-	×
Properties Information		
Type of rounding		
Round to decimals		~
Number to round		
DecimalValue		~
Digits to round		
3		
Peturn value		
Reunding/Jule		
E	Save	Close
Activity properties		

Possible rounding options

### <u>Ceil</u>

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

### <u>Trunc</u>

Drops everething 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

# 6.15 Rest operation

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

Uses <u>System.Net.HttpClientComponent.TNetHTTPClient</u>

S Edit properties - REST op	eration		-		×
Properties Information					
Operation	Content-Type				
GET	~				
URL				A	
'http://worldtimeapi.o	org/api/timezone/Europe/Am	ısterdam'			
Body Parameters Authe	ntication Connection				
Body				A	
Operation result type					
Content as string value	⊖ HttpResponse entity				
Return value					
HttpResponse					
			Save	→ CI	ose

Activity poperties

```
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 <</pre>
300) then
      Response := HttpResponse.ContentAsString
    else
      raise ENetException.CreateFmt('%d %s%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. <u>Click here</u> 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.						
В	ody	Parameters	Authe	entication	Connection		
C	Ad	d E	dit	Dele	te		
	Kind			Name		Value	
Þ	URL-S	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

## HttpResponse

As of codolex Version 2.4.0 202 its also possible to recieve the result as an HttpResponse Entity.

This entity is added as plugin datasource [43] entity by default in codolex.



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

REST operation	 Create StatusCode variable	 Get by association
HttpResponse	Integer	List < HttpHeaderValue>

# 6.16 Regular expressions

The reg-ex activities make it possible to do advanced search and replace actions on strings

🗸 🔚 Re	gular expressions
	Escape chars (RegEx)
	ls match (RegEx)
-0	Replace match (RegEx)
	Search match (RegEx)
-0	Split string (RegEx)

More about the options available in reg-ex: Options [142]

Escape chars 42 IsMatch 43 Split string 44 Search match 45 Replace match 46

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?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]\*)\*\/?\$
- 5. Extracting file extension:  $\[0-9a-z]+\$

### 6.16.1 Options

Most of the reg-ex activities have the following options possible

options	
Compiled	None None
Explicit capture	Not empty
Ignore case	Single line
Ignore pattern space	
Multi line	

System.RegularExpressions.TRegExOption

### 6.16.2 Escape chars

Use the 'Escape chars' activity to replace special characters with their escape codes.

Uses <u>System.RegularExpressions.TRegEx.Escape</u>

S Edit properties - Escape chars (RegEx)		-	$\times$
Properties Information			
String value			
StringValue			~
Use wildcards			
Return value			
EscapedStringValue			
,			
	B	Save	Close

Activity properties

```
begin
  var EscapedStringValue: string;
  EscapedStringValue := TRegEx.Escape(StringValue, True);
end;
```



If 'Use wildcards' is checked, the '\\*' or '\?' characters are not converted.

### 6.16.3 IsMatch

The 'Ismatch' activity returns a boolean if a regex pattern returns a match on a string

Uses <u>System.RegularExpressions.TRegEx.IsMatch</u>

S Edit properties - Is match (RegEx)		_	×
Properties Information			
RegEx pattern			
RegexPattern			~
Match pattern			
StringValue			~
Options			
Compiled Explicit capture Ignore case Ignore pattern space Multi line	<ul> <li>None</li> <li>Not empty</li> <li>Single line</li> </ul>		
Return value			
lsRegexMatch			
		Save	Close
Activity properties			

begin	
<pre>var IsRegexMatch: Boolean;</pre>	
<pre>IsRegexMatch := TRegEx.IsMatch(StringValue, RegexPattern, []</pre>	);
end;	

Resulting Code

# 6.16.4 Split string

Split string splits the string on rexeg matches and returns as a codolex string list.

Uses <u>System.RegularExpressions.TRegEx.Split</u>
🔇 Edit properties - Split string (RegEx)		_		×
Properties Information				
RegEx pattern				
RegexPattern				~
Split value				
StringValue				~
Options				
Compiled Explicit capture Ignore case Ignore pattern space Multi line	<ul> <li>None</li> <li>Not empty</li> <li>Single line</li> </ul>			
Return value				
SplitedRegex				
		Save	•	Close

Activity properties

```
begin
var SplitedRegex: ICodolexList<string>;
var TextArray := TRegEx.Split(StringValue, RegexPattern, []);
SplitedRegex := TCodolexList<string>.Create;
SplitedRegex.AddRange(TextArray);
end;
```

Resulting Code

# 6.16.5 Search match

The 'search match' activity searches for a match in a string for a reg-ex pattern and returns the match. Uses <u>System.RegularExpressions.TRegEx.Match</u>

# Activities

Sedit properties - Search match (RegEx)		-		×
Properties Information				
RegEx pattern				
RegexPattern				~
Match pattern				
StringValue				~
Options				
<ul> <li>None</li> <li>Ignore case</li> <li>Multi line</li> <li>Explicit capture</li> <li>Compiled</li> <li>Single line</li> </ul>	<ul> <li>☐ Ignore pattern space</li> <li>☐ Not empty</li> </ul>			
Returns				
• Single match O Match collection				
Return value				
MatchResult				
		Save	-	Close
Activity properties				
<pre>begin var MatchResult: TMatch; MatchResult := TRegEx.Match(String)</pre>	gValue, RegexPatte	ern, []);		



The activity returns an object of the **TMatch** variable. If the option 'Match collection' is chosen, the result will be a **TMatchCollection** 

# 6.16.6 Replace match

The 'Replace match' replaces the matched part of a string to the replacement, and returns the new total as result. *Uses <u>System.RegularExpressions.TRegEx.Replace</u>* 

S Edit pr	operties - Replace match (RegEx)		—		$\times$
Properties	Information				
RegEx patt	ern				
RegexPatt	ern				~
Match patt	tern				
StringValu	e				~
Replaceme	ent				
Replacem	ent				~
Options					
Comp Explic Ignore Ignore Multi	iiled it capture e case e pattern space line	<ul> <li>None</li> <li>Not empty</li> <li>Single line</li> </ul>			
Return valu	Je				
Variable					
			ave	₽	Close
Activity	properties				

```
begin
  var Replacement: string;
  var Variable: string;
  Variable := TRegex.Replace(StringValue, RegexPattern, Replacement,
[]);
end;
```



# 6.17 String utils

The string utils sections offers most of the stringutils functions available in delphi.



# 6.17.1 String change

The string change activity offers the delphi stringutils functions to change a string.

🔇 Edit pr	operties - String change		-		$\times$
Properties	Information				
Function to	o apply				
Quoted					~
Input string	g value				
StringValu	e				~
Return valu	ie				
Quote	edString				
		_			
			Save	->	Close

Activity properties

Retrurn 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.*"

# <u>Join</u>

Joins an array string with a separator.

```
Example:
Array = ['the', 'quick', 'brown'];
Separator = -
Position = 1
Number of ietms = 2
```

```
begin
  var ChangedString: string;
  var ArrayToJoin := StringArray;
  ChangedString := String.Join('-', ArrayToJoin, 1, 2);
end;
```

*Result* = "quick-brown"

Separator is optional Postion is optional with a default of 0 Number of items is option 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."

# 6.17.2 String check

The string check activity can be used to validate various things about strings

S Edit pr	operties - String chee	:k		-	×
Properties	Information				
String chec	k function				
Equals					~
Input string	j value				
StringToCl	neck				~
String value	e to compare				
StringToCo	ompare				~
✓ Is case s	ensitive				
Return valu	e				
IsSameStri	ng				
				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 := EndsStr(StringToCompare, StringToCheck);
end;
```

Result = True

# Is empty

Example: StringToCheck = "Hello World"

```
begin
  var var IsSameString: Boolean;
  IsSameString := StringToCheck.Trim.IsEmpty;
end;
```

Result = False

# 6.17.3 String conversion

The string conversion activity can be used to convert a string to another variable type or to upper/lower case the string.

S Edit properties - String conversion		—		$\times$
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

# **<u>Bigint</u>** (Int64)

Example: String = "34214"

begin

```
155
```

```
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* 

# 6.17.4 String find

The string find activity can be used to find the length or position of a sub string in a string



Activity properties

begin
<pre>var StringLength: Integer;</pre>
<pre>StringLength := StringToFind.Length;</pre>
end;

Resulting Code

# **String length**

String length simply returns the length of the given string with the <u>length</u> 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.

# 6.17.5 String parts

The string parts activity can be used to get different parts of the activity

S Edit properties - String parts	_		×
Properties Information			
Apply string part function			
Before text			~
Input string value			
StringToGetPortOf			~
Text to search for		4	
'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);
```

Result: " brown ".

# **Right part**

end;

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 -> <u>Right part</u> & <u>Left part</u>

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".

# 6.17.6 String split

The string split activity can be used to split strings on a delimiter and return the result as array

🔇 Edit properties - String split	-		×
Properties Information			
String value			
StringToSplit			~
Delimiter			
Return value			
SplitStringArray			
	Save	→	Close
Activity properties			

<pre>begin var SplitStringArray: ICodolexList<string>; SplitStringArray := TCodolexCollections.CreateList<string>; var SplittedParts := SplitString(StringToSplit, ',');</string></string></pre>
<pre>SplitStringArray.AddRange(SplittedParts); end;</pre>
Resulting Code

# 6.18 Variables

The variable activities can be used to create entities/variables and manipulate them.



List Operations 165 Copy entity data 167 Free Object 168 Get by association 169

# 6.18.1 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.

S Edit pr	operties - Create variable		-		$\times$
Properties	Information				
Scope					
O Declare	local variable	🔘 Initialise class variable			
Data type					
Array					~
Initializatio	n			A	
Exisitin	ngStringArray				
Collection	type				
String					$\sim$
Return valu	e				
StringArray	/				
			Save	<b>→</b> (	Close

Activity properties

```
begin
var StringArray: TArray<string>;
StringArray := ExisitingStringArray;
end;
```



Possible variable types:

- Array -> TArray<T>
- Bigint -> Int64
- Binary -> ICodolexBinary
- Boolean -> Boolean
- Custom
- Date/Time -> TDateTime
- Decimal -> Double

# Activities

- Entity
- Enumeration -> unsuported
- 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.

Data type



# 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.

Data type		
Entity		~
Entity		
Codolex.Shippers		
Properties		Q
Name	Туре	Value
CompanyName	String	'CreatedCompany'
Phone	String	
Orders_ShipVia_ShipperIDLis	List <orders></orders>	OrdersList $\lor$

# Initialize class variable

When using class variables in a flow class, the initialisation is possible in every **non-class** flow of that class.

S Edit pro	perties - Creat	te variable		-		×
Properties	Information					
Scope						
O Declare I	ocal variable		<ul> <li>Initialise class variable</li> </ul>			
Variable						
– Class: Flo	wClassString					~
Initialization	ı					
'Initial	ised String	:				
				Save	•	Close

# 6.18.2 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.

S Edit properties - Char	-	×		
Properties Information				
Variable to change				
Shippers				~
Properties				ø
Name	Туре	Value		
CompanyName	String	'NameChange'		
Phone	String			
Orders_ShipVia_Shipper	List <orders></orders>	OrdersList		
			Save	Close

Activity properties

```
begin
  Shippers.CompanyName := 'NameChange';
  Shippers.Orders_ShipVia_ShipperIDList := OrdersList;
end;
```

Resulting Code

# 6.18.3 List Operations

Use the list operation activity to control list-related tasks. Uses <u>System.Generics.Collections.TList</u>



Activity properties

```
begin
var GreaterStrings: ICodolexList<string>;
GreaterStrings := StringList.Filter(
function(Value: string): Boolean
begin
        Result := Value.Length > 10;
end
);
end;
```



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.

C Edit properties		_	×
Properties			
Description			
Integer with most digits in Collection			
Variable			
Collection			~
List operation			
Max			~
Comparison expression ("Left" and "Right" are collection element variables)			÷
Left.ToString.Length - Right.ToString.Lengt	:h		
Return value			
Element			
		<u>S</u> ave	Close

# 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.

# 6.18.4 Copy entity data

The copy entity data activity can be used to transfer the values of an instance of an entity to another variable.

S Edit properties - Copy entity data		—		×
Properties Information				
Copy values from				
FromEntity				~
To variable				
ToEntity				~
Exclude fields				
CompanyName				
	💾 Sav	e	₽	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.

# 6.18.5 Free Object

The Free object can be used to release the contents of a variable. Uses <u>System.SysUtils.FreeAndNil</u>

S Edit properties - Free object	_	×
Properties Information		
Variable to free		
FreeAbleObject		~
	Save	Close
Activity properties	Jave	CIUSE
Activity properties		
begin		

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.

# 6.18.6 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.

Edit properties				-	
Properties					
Name					
Customer					
Database tablename					
Customer					
Fields Associations					
🔒 <u>N</u> ew 🔀 <u>D</u> elete					
FromEntity	FromField	Name	AssociationType	ToEntity	4
> Customer	Companyld	Company	OneToMany	Company	
Invoices	CustomerId	FK_0	OneToMany	Customer	
<					►
				Course 1	

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:

C Edit properties	-		×
Properties			
Description			
Get the company from the customer			
Variable			
Customer			~
Assocation			
Company			~
Return value			
Company			
	<u>S</u> ave	₽	Close

# 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

# 6.19 Zipping

The zipping activities allows for easy creation and extraction of zip files.

Zipping
 Add file to zip
 Close zip
 Create/open zipfile
 Extract zipfile
 Listing zip
 Read zip

Create/open zip file
Extract zipfile 172
Add to zipfile 173
Close zipfile
Listing zip 175
Read zip 176

# 6.19.1 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 <u>System.Zip.TZipFile</u>

S Edit pr	operties - Create/o	pen zipfile			-		×
Properties	Information						
Path to Zip	file						
ZipFilePat	ı						~
Return valu	e						
ZipFile							
				📋 s	ave	<b>→</b>	Close
Activity	properties						
hegin							

```
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.

# 6.19.2 Extract zipfile

Extract zipfile can be used to extract files from a zip, and place it at a destination

Uses System.Zip.TZipFile.Extract

S Edit properties - Extract zipfile	-		×
Properties Information			
Zip file or path to zip file			
ZipFile			~
Extract destination			
Destination			~
File or index to extract (optional)			
ZippedFile			
Create sub folder			
	Save	→	Close

# 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 <u>Close the zipfile</u> after extracting from the zip.

# 6.19.3 Add to zipfile

Add to zipfile can be used to add files or directories to an open zip file. *Uses <u>System.Zip.TZipFile.Add</u>* 

S Edit pr	operties - Add file to zip		—		$\times$
Properties	Information				
Zip file					
ZipFile					~
File(s)/con	tent to zip				
StringArra	у				~
Compressi	on type				
Deflated					~
< Add all	files from folders				
🖂 Include	subfolders				
			Save	₽	Close
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 wikki: <u>System.Zip.TZipCompression</u>

When adding folders, there are 2 options to concider. 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 <u>Close the zipfile</u> after adding to the zip.

# 6.19.4 Close zipfile

The close ZipFile must be used after extracting or adding data to a .zip file Uses <u>System.Zip.TZipFile.Close</u>

S Edit pro	operties - Close	e zip		_		$\times$
Properties	Information					
Zip file						
ZipFile						~
				Save	•	Close
A . 1 · · ·						

Activity properties

begin
ZipFile.Close;
end;

Resulting Code

# 6.19.5 Listing zip

'Listing zip' can be used to list all the files that are present in the zip file. *Uses <u>System.Zip.TZipFile.Read</u>* 

S Edit pro	operties - Listing zi	D			_	. 🗆	×
Properties	Information						
Zip file or p	ath to zip file						
ZipFileToU	se						~
Listing con	tent						
Both							~
File index in	n zip archive						
Return valu	e						
ListedZip							
				_			
					Save		Close

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.

# 6.19.6 Read zip

The read zip activity can be used to read the contents of a file in a zipfile without extracting.

S Edit properties - Read zip	_	×
Properties Information		
Zip file or path to zip file		
ZipFileToUse		~
File name or index in archive		
FileName		
Output to		
CustomArray		~
And the supervise	Save	Close
ACTIVITY properties		
<pre>begin ZipFileToUse.Read('FileName', CustomArray); end;</pre>		

Resulting Code

The output must be a variable of type TBytes, TStream or TArray<Byte>.

# 6.20 Advanced

Enter topic text here.

# Creating your own activity

# 7 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".

# 7.1 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 <u>SynEdit</u> 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.

All configurations - All platforms	✓ <u>A</u> pply <u>Save</u>
Conditional defines	
DCP output directory	.\Bin\Dcp\\$(Platform)\\$(Config)
Framework search path	
Package output directory	.\Bin\Bpl\\$(Platform)\\$(Config)
Search path	.\Libraries\\$(Platform)\\$(Config)\
Show general messages	false



# **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 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.

> Delphi Compiler				
> Resource Compiler	Target			
Build Events	Debug configuration - Windows 32-bit platform	~	Apply	<u>S</u> ave
Application     Version Info     Packages	De <u>s</u> cription			
Debugger	Library name			
Symbol Tables	LIB prefix:			
	LIB suffix: <u>S(Auto)</u> LIB <u>v</u> ersion:			 
	Usage options	Build control		
	O Designtime only	Rebuild as needed		
	• Runtime only	○ Explicit rebuild		
	O Designtime and runtime			

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"

```
Image: Section TCodolexPlaySoundPlugin.Components: IList<IFlowPluggableComponent>;
    begin
    Result := TCollections.CreateList<IFlowPluggableComponent>;
    var PlaySoundActivity := IPlaySoundPluggableComponent.Create;
    Result.Add(PlaySoundActivity);
    var StopSoundActivity := IStopSoundPluggableComponent.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.

## 7.2 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".

#### ∼ 🚞 PlaySound

Codolex.Activity.PlaySound.CodeGen.Delphi.pas (Codole...

Codolex.Activity.PlaySound.Component.Interfaces.pas

Codolex.Activity.PlaySound.Component.pas

- > 🖹 Codolex.Activity.PlaySound.Editor.pas
  - Codolex.Activity.PlaySound.Pluggable.pas
  - Codolex.Activity.PlaySound.Storage.pas
  - Codolex.Activity.PlaySound.Validator.pas

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, IF
public
  function Name: string;
  function Description: string;
  function Category: string;
  function CreateComponent(const Flow: IFlow): IFlowComponent;
  function CodeGenerator(const FlowComponent: IFlowComponent): IFlowPluggableComponent
  function Storage(const Container: TContainer): IFlowPluggableComponentStorage;
  function ShowEditor(const FlowComponent: IFlowComponent; const Container: TContainer);
  function Validator(const FlowComponent: IFlowComponent; const Container: TContainer;
  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 IFlowVariable and PlayMode of the type String

```
uses
FlowModel.Activity.Interfaces,
FlowModel.Variable.Interfaces;

type
IFlowActivityPlaySound = interface(IFlowActivity)
['{C93C952C-4268-4506-B64C-AD988F49A750}']
function GetSoundFile: IFlowVariable;
function GetPlayMode: string;
procedure SetSoundFile(const Value: IFlowVariable);
procedure SetPlayMode(const Value: string);
property SoundFile: IFlowVariable 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 IFlowVariable should be ignored by the JSONMarshaller in the storage because we need to serialize the references ourselves, so we add the [JSONMarshalled(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
[JSONMarshalled(False)]
FSoundFile: IFlowVariable;
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;

With the inherited function AfterConstruction it's also possible to add default values to the properties defined in the interface.

With the inherited function CreateActivityVariable, 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

```
ReturnVariable.VariableType := TDataType.Custom_;
ReturnVariable.CustomTypeName := 'TYourCustomType';
```

## **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 Codolex 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 IFlowVariable. 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.

```
Codolex.Activity.SoundFile.Component.Interfaces,
Codolex.Modelling.Helper.Interfaces,
```

```
FlowModel.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.

2. Create class properties for the FlowVariable and the FlowVariable input.

#### Private

```
FSoundFile :IFlowVariable;
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: IFlowVariable);
...
procedure TfrmActivityPlaySound.OnSoundFileChange(Variable: IFlowVariable);
begin
    FActivity.SoundFile := 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: IFlowVariable
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: IFlowVariable
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,
    FActivity,
    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.OnSoundFileChange(Variable: IFlowVariable);
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 "JSONMarshalled False".

```
procedure TActivityPlaySoundSerializer.DoAfterSerialize;
var
Activity: IFlowActivityPlaySound;
begin
inherited;
Activity := FEntity;
SerializeReference<IFlowVariable>(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: IFlowElement)
        begin
        Activity.SoundFile := Variable as IFlowVariable;
        end);
end;
```

## 7.3 Tags

Earlier in this article, while discussing the Codolex.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 Codolex. 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 four

```
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
var
Activity: IFlowActivityPlaySound;
begin
if not Supports(FlowComponent, IFlowActivityPlaySound, Activity) then
raise EInvalidOpException.CreateFmt(ValidationNotAvailableForActivity, [FlowCor
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.

## 7.4 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: ISourceCode);
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.

190

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 SourcCode 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, Codolex uses IncreaseIndent/DecreaseIndent to places 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 ifstatements 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'.

#### .Variabele

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.of.enum')
```

.add('::TempParamName');

will result in

uses
 Unit.of.enum
..
var TempEnumVar: TMyOwnEnum;

on the place where the param was added.

#### .AddDependancy

Speaking of dependencies, you can also use the .AddDependancy function to add "Unit.of.enum". By default, this use is added in the implementation section. If you want it in the interface, use:

.AddDependancy('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', 'PWideChar')
.Add('::SoundFileChar')
.add('::SoundFileChar := PWideChar(::SoundFile);')
.Add('sndPlaySound(::SoundFileChar, ::PlayMode);')
.end_;
```

## 7.5 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 Codolex.Activity.YourActivityName.Validator file. This file provides a DoValidate function to Codolex 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.

#### .AddWaring

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.Core.PlaySound'}
```

# **Command line interface**

## 8 Command line interface

It is possible to invoke code generation via the command line application. This allows Codolex to be integrated with a build server or with continuous integration. CodolexCLI.exe tool can be found in the installation directory of Codolex, and has following commands available:

-help : Show help information -build <filepath>: Build the given Codolex project

# **Best practices**

## 9 Best practices

## 9.1 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.

## 9.2 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

## 9.3 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 [54] page for more information

# FAQ

**Question**: I downloaded Codolex. When I try to start the program, an error with the message: The procedure entry point @{...}@{...} could not be located in the dynamic link library {...}/CodolexSynEditDR2\*0.bpl. Why do I get this error? And how do I solve it?

**Answer**: Codolex is always built for the latest patch for any Delphi version. The libraries that Codolex uses are based on the procedures of Delphi. That means that if Codolex is built for Delphi 11.3, there could be a mismatch in functions when installing the tool on Delphi 11.1. To solve this error, it's recommended to update Delphi to the latest version/patch available. Be sure to check out the <u>wiki from Embarcadero</u> for the latest version.

**Question**: When I start using a flow in my own source code, I get the message that the Codolex.Collections is not found.

**Answer**: You may get this message if Codolex is not properly configured in the Delphi library path. On your system, find where the

Codolex.Collections.dcu is located. By default, it is the %APPDATA% \Codolex\Release\22.0\DCU folder.

Add respectively the Win32 folder to the Win32 library path and the Win64 folder to the Win64 library path.

> IDE > User Interface > Language Toxicity Metrics Library Selected Platform Windows 32-bit
V Delphi
Library     Library ath       Library - Translated     Image: State at the stat the stat the state at the state at the state at the state at th
> Formatter     S(BDSCOMMONDIR)\Bpl        > Version Control     DCP output directory        > Deployment     S(BDSCOMMONDIR)\Dcp        > Translation Tools     Brownian path
> Modeling     browship path       > Debugger     \$(BDS)\OCX\Servers;\$(BDS)\SOURCE\VCL;\$(BDS)\source\ttl\common;\$(BD5 \square)       Unit scope names
Debug DCU path S(BDSLIB)\\$(Platform)\debug HPP output directory
S(BDSCOMMONDIR)\hpp\S(Platform) Save Cancel Help

Question: When I run my application, I get the exception ECodolexDataSetConfig: "Databroker not found for class". Answer: Add the following compiler directive to your Delphi project, in the (for example) DPR file: {\$TRONGLINKTYPES ON} Codolex uses RTTI information to retrieve the Databroker for entities dynamically. See <u>Docwiki of Embarcadero</u> for more information.

# **Release notes**

### 11 Release notes

Enter topic text here.

## 11.1 Version 2.4.0

# Highlights

## HttpResponse for Rest activity

S Edit properties										
Properties										
Name										
HttpResponse										
Fields Associations										
Name	DataType	Size	Primary key	ls collection						
HttpResponseld	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 codolex 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 HttpHeaderValue list.



## 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 trough 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.

#### 11.1.1 Version 2.4.1

This version is a fix for version 2.4.0

## **Bugfixes**

We fixed an error that could arise when the <u>DMVCFramework</u> was installed. It could confilct on some units that were contained in Codolex framework and the DMVC framework.

The conflicting files are not included in the Codolex framework anymore.

#### 11.2 Version 2.3.0

## **Highlights**

#### **Al 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 recieve 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.

C Edit properties - Al Chat		_			×
Properties Information					
Description					
Authorization key				A	<b>7</b> 7
Model					
GPT 4					×
List of all chat messages					
					~
Content				A	"
Return value					
ChatMessage					
	_				_
		<u>S</u> ave	-	Clo	se

# **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.

Edit properties - Create varia	ble			-		×
Properties Information						
Scope						
Declare local variable	🔘 Initialise	class variable				
Data type						
Entity						~
Entity						
Codolex.Orders						
Properties						Q
Name	Туре	Value				-
ShipCountry	String					
Order Details_OrderID_Order	List <order details=""></order>	OrderDetailsList				$\sim$
Shippers_ShipperID_ShipVia	Shippers	OrderDetailsList				
Customers_CustomerID_Cus	Customers					Ŧ
Initialization						<b>A</b> 75
Return value						
Orders						
			_		_	
				<u>S</u> ave	-	Clo

## Addition of data type JSONValue

Working with JSON got much better in Codolex 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.

Edit properties										-		×
Properties												
Name												
EntityWithJSONValue	EntityWithJSONValue											
Fields Associations	Fields Associations											
🔎 <u>S</u> earch usage 🗋 <u>N</u> ew 🗙	D <u>e</u> lete <u>Up</u> <u>D</u> own											
Name	DataType	Size	Primary key	Is collection								
ChildObject	JSONValue	1	False									

# **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:

• <u>Custom database connections</u> 37

> - Core.Structural.DataSource

• <u>Nexus DB</u> 37

## 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

## 11.3 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

Flowclass variable properties Properties	_		×
Variable name			
Data type			
Bigint Binary Boolean Custom Date / time Decimal Entity Enumeration FlowClass Integer List String			1
Sa Sa	ave	× Ca	ancel

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.

€ Edit properties - □ ×
Properties Information
Description
Scope
O Declare local variable
Data type
~
Initialization
I
Return value
Variable_1
Save Save

## **Other features**

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.

	€ Edit properties - □	2
	Properties Information	
	Description	
	Get amount of courses	
	Database command to execute	
	'SELECT COUNT(*) as CourseAmount FROM Course'	
	Datasource to run database command	
Execute command	DefaultDB	
🕈 🖗 String	Result field data type	
CourseAmount	String	
	Result field name	
	CourseAmount	
	Return value CourseAmount	
	💾 Save 🕞 Clo	ose

## List as asstribute

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
	menulD	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 expension 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.-

## 11.4 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.