ConceptDraw DIAGRAM Third Party Developer’s Guide

CS Odessa corp.

CONTENTS

1. ConceptDraw DIAGRAM for Developers.

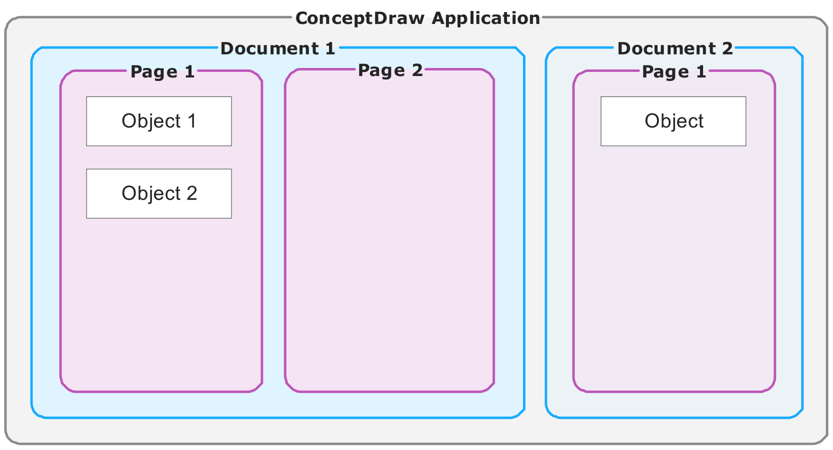
Introduction

This document is the current version of the developer documentation. The document has the following structure:
Chapter #2 represents the general information on the data structure in ConceptDraw and methods for their processing.
Chapter #3 describes the types of objects available for creating new solutions by the end users.
Chapter #4 contains the list of common tasks and solutions’ scenarios.
Chapter #5 describes the data management techniques.
Appendix contains the relevant reference material on developer tools.

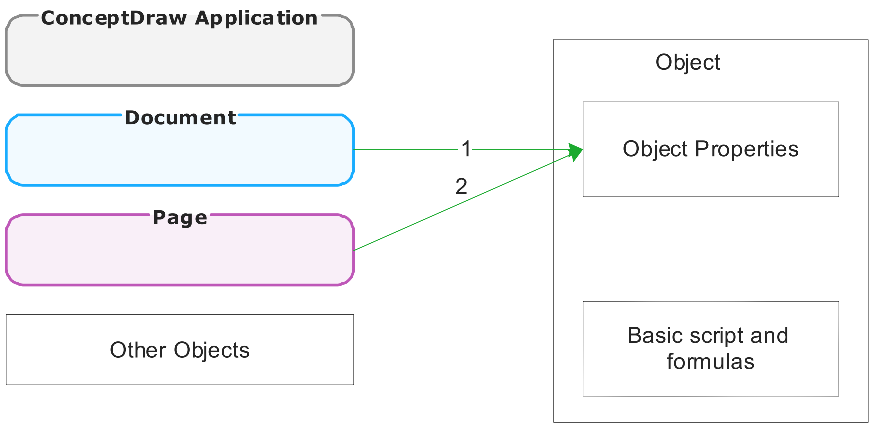
2. Developer Tools

2.1 The data exchange structure in ConceptDraw DIAGRAM.

The ConceptDraw DIAGRAM deals with processing of the user documents that can consist from the number of pages containing the graphic objects.



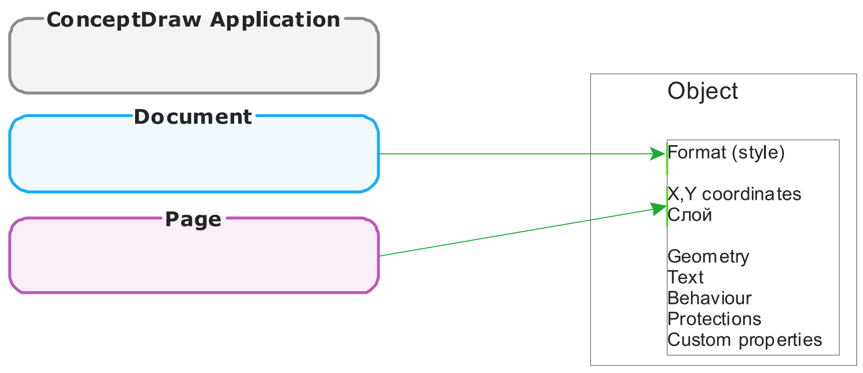
Each graphic object has a set of parameters (properties) that can be changed using interface as well as using custom data via supported API



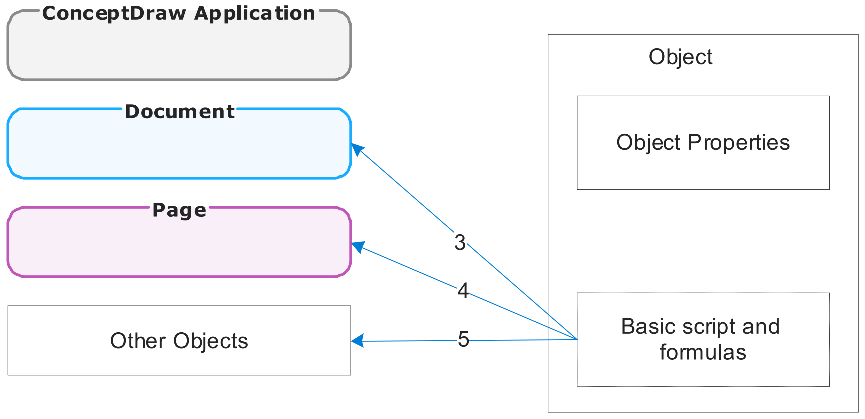
Each graphic object can use the predefined document styles. Thus, the document may "influence" on the object (1). The objects are represented in the pages of the document and have the coordinates of the location (2) and may be on different logical layers.

Note: The application has two separate, unrelated entities:

1) *Layer - the logical layer, which is controlled from a dialogue "Layers", and is not associated with the visual stacking order of objects.*
2) *Unnamed visual level of objects' overlay, separate for each object, which is regulated by the commands StepFront, StepBack, SendtoFront, SendtoBack.*



Graphical objects can have a special script that allows changing the document properties (name, page size, settings). Also it can allows managing the page (name, order, background) as well as, managing other objects (create, delete, format) - these relations are shown in Figure ## 3-4-5.



2.2. Data management in ConceptDraw DIAGRAM

Each ConceptDraw graphical object has a set of properties that define its view and functionality in the application. ConceptDraw offers 3 ways to access properties of objects: a table of properties, the script, XML. The table below represents the groups of objects' properties and the ability to access them by using different API.

Object properties	**Table of properties**	**Basic script**	**CD XML**
Placement and size	Yes	Yes	Yes
Layer	**No**	Yes	Yes
Object formatting (line, fill and shadow)	Yes	Yes	Yes
The geometry (contour of the shape)	Yes	Yes	Yes
Text and formatting	Yes	Yes	Yes
Blocking	Yes	Yes	Yes
Custom behavior	Yes	Yes	Yes
Hyperlink	**No**	Yes	Yes
Custom properties	Yes	Yes	Yes
Data sources	Yes	Yes	Yes
Processing sets of variable length	No	Yes	No

2.3 Developer Tools

ConceptDraw DIAGRAM provides three ways to manage the graphic content of their documents:

- Shape parameters table

- ConceptDraw Basic Script

- ConceptDraw XML.

With the help of properties table one can define the logic of the objects’ behavior, apply them the arbitrary appearance and set the connection between the appearances of different objects as well as create complex objects. The object properties table gives the opportunity to work with the already defined set of parameters and properties of an object. It is designed to manage the limited (including large) set of properties. CDBasic Script helps to interact with other applications (command line running, invocation of the custom functions from external libraries).

ConceptDraw Basic Script allows you to create and work with more complex objects, the objects with a variable number of child elements, and custom properties, as well as pre-defined behavior. CDBasic Script is only useful for such tasks.

ConceptDraw XML allows third-party applications to create graphic documents “on the fly” and generate them by means of special simplifying constructions.

2.3.1. The principles of using a table and CDBasic in a particular situation

When it is better to work with tables, and when with CDBasic? All that can be done using a table (which is not so much: actually everything that can be defined as a change in the properties of the Shape and its parent or child Shape) is carried out faster exactly through table. But if you need to draw, or to modify properties of a graphic object or other ConceptDraw objects (e.g., document, application layer e t c), or just needed more advanced features provided by Basic, you should use CDBasic.

Be note that you should avoid situations in which the Basic script processing depends on the table parameter that are calculating at the same time, because you cannot determine in advance the sequence of calculation, and when any procedure called from a table variable will be invoked . In such cases it is better to do everything through CDBasic.

And finally, there are cases where it is justified in the interaction tables and CDBasic.

3. Objects

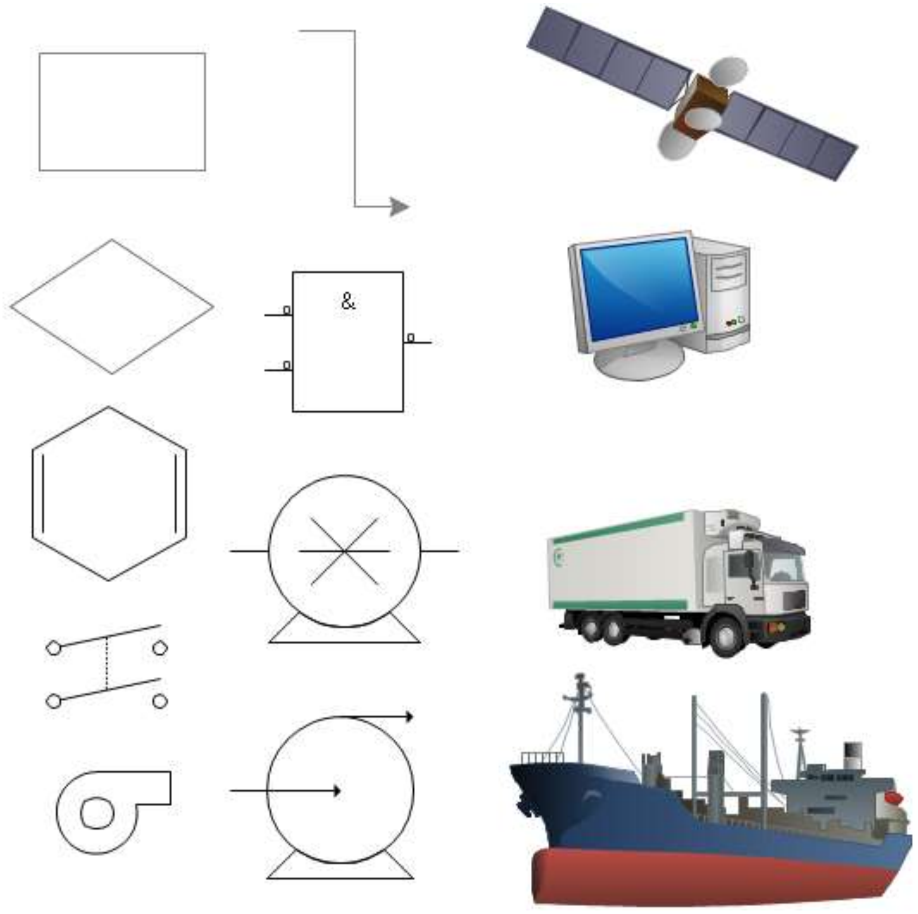
ConceptDraw DIAGRAM allows you to create and work with several types of graphical objects - from simple images to intelligent objects changing its appearance depending on external data (Smart Objects).

3.1. Simple Drawing objects

Simple Drawings are the simple graphic objects created with the drawing tools and primitives grouped into more complex objects that are simply have a set of graphical and text properties. The behavior of the objects (size, position, styles and formatting) is taken in the usual application method.

The behavior of such objects is similar to the behavior of objects in other graphic applications,

The objects are not interactive. These objects are created by a simple drawing and combining of simple objects into groups.



You can create your own set of simple objects; store them into the libraries and create templates with predefined settings.

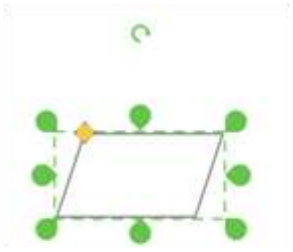
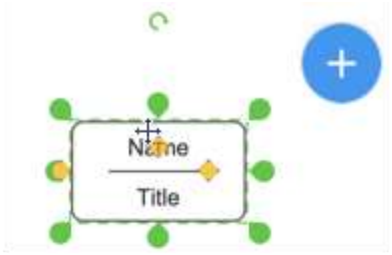
Such objects have no logic and are vector images that can be used for creating static illustrations. They have the usual set of data inherent to the graphical objects.

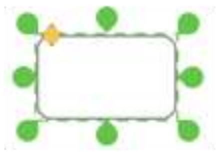
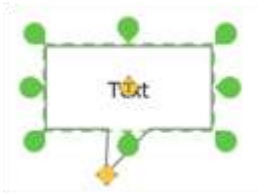
3.2. Special Drawing objects

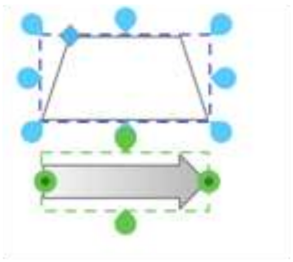
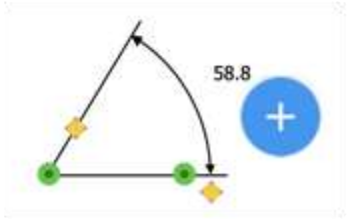
Special Drawings - are the special objects that support the special settings for some properties. Such object does not behave in a standard way even during common operations.

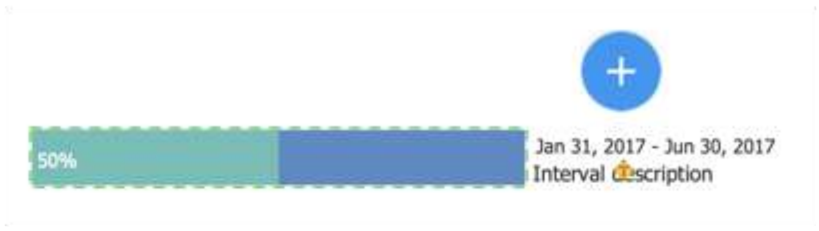
This category includes objects that behave in predetermined manner when properties changes (for example, a disproportionate increasing of object's parts during resizing). Also it includes the objects protected from properties changes, and objects, the type and state of which may be changed using the control points.

Special Drawing object can be created using the blocking of certain object's properties (the most simple objects) or by setting some dependencies on other properties of the object. In the second case, the behavior of such objects is determined in the Shape Parameters table.



Appearance and behavior of the object can depened on the random custom control points - Control Handles. Such objects can be created using the object properties table.

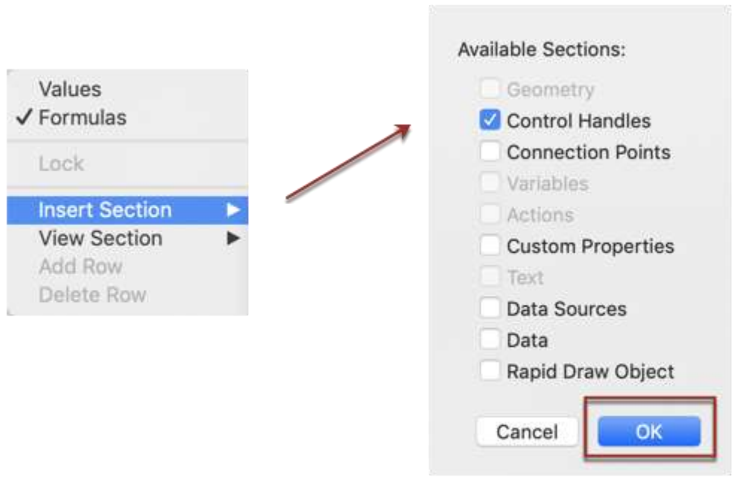
This group includes objects with locked properties (such as moving the X-axis, changing the width of the object, etc.), as well as objects with non-trivial behavior when changing their properties with custom control points.

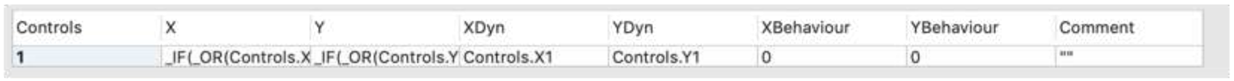
These objects are created using the blocking properties, installation of special relationships between the properties, as well as by adding a custom control points

These objects are used in more complex drawings, when customization of the existing graphic objects or interactivity in diagrams required.

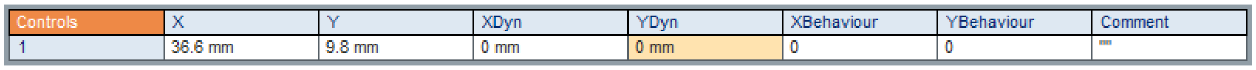
Setting of such objects is carried out in the object properties table. Depending on the desired behavior of the object any relationships between the properties can be established.

To create an object with a custom control points you need to add the Control Handles section into the Shape Parameters table.





Controls	X	Y	XDyn	YDyn	XBehaviour	YBehaviour	Comment
1	_IF(_OR(Controls.X	_IF(_OR(Controls.Y	Controls.X1	Controls.Y1	0	0	""



Controls	X	Y	XDyn	YDyn	XBehaviour	YBehaviour	Comment
1	36.6 mm	9.8 mm	0 mm	0 mm	0	0	""

Each control point has X and Y parameters that determine coordinates of the point within object. Also other properties of an object may depend on these coordinates. To move a control point only along one axis, you can use the fields XBehaviour and YBehaviour. By setting one of these properties, a value 1, you can restrict the movement of the control point by one axis.

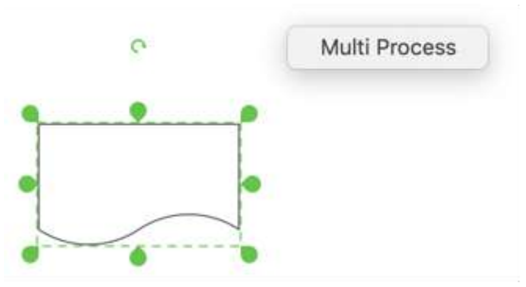
The objects containing special relationships between the graphical properties can also contain the custom control points.

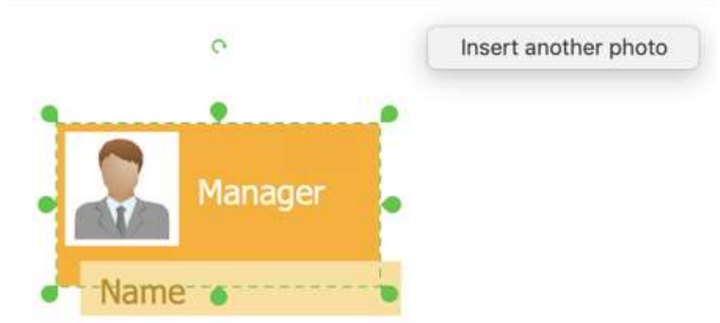
3.3. State drawing objects

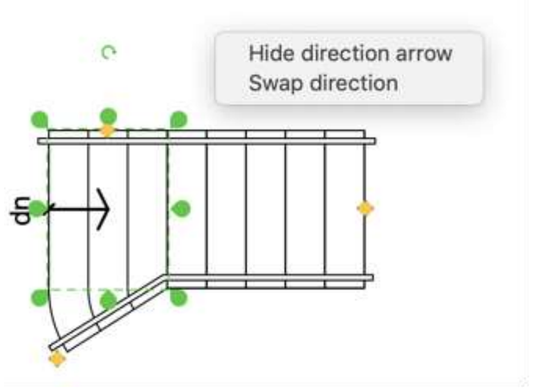
State Drawing (switchable objects) - objects with multiple final states, which change their state to implement a predefined command by the user.

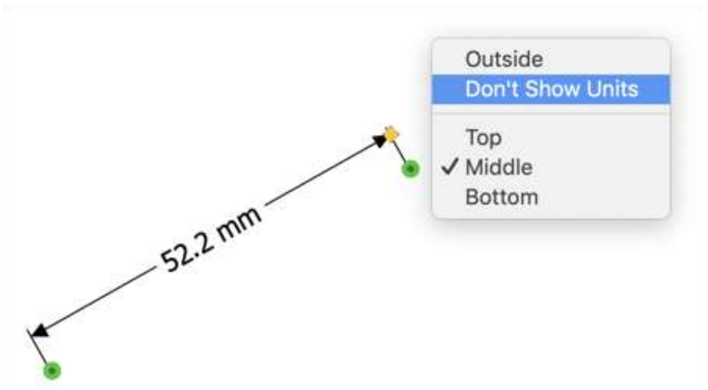
Typically, such objects can On / Off such features, as visibility of the certain parts; color changes or changes of the object's shape/outline as well as its internal objects. Also, these objects can change their view by the built-in function (Action), such as cyclic: rectangle, rhombus and parallelogram view. And also it can be the interface elements, for example callouts that can have one of the predefined forms.

Every such object has several predefined commands – Actions. Each command includes a certain logic state. And the object changes its appearance according to this state.



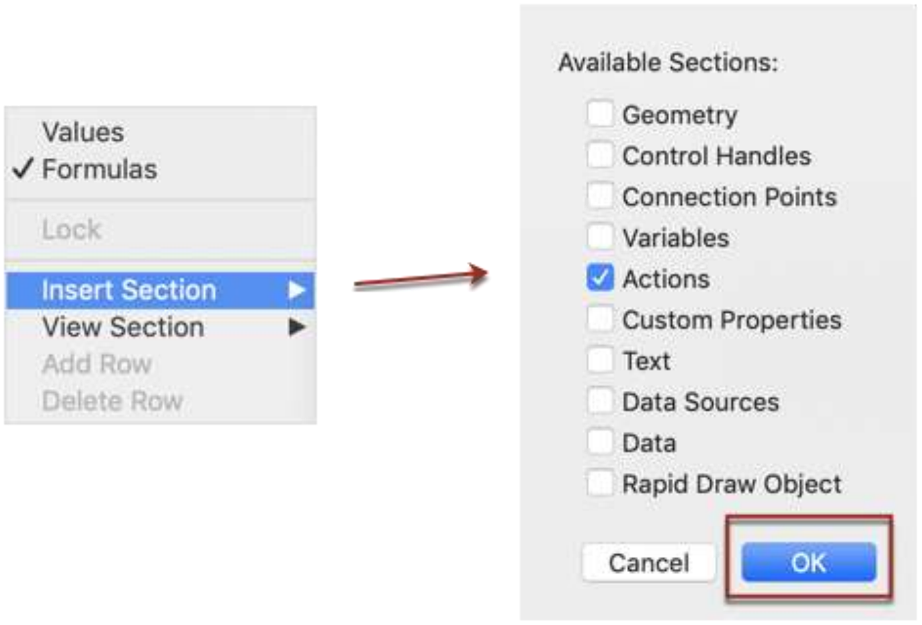


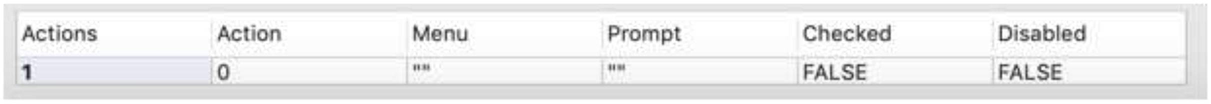




To create the State Drawing objects, you must understand the fundamentals of programming and have a basic knowledge of object properties table. The custom commands – Actions have to be created for such object.

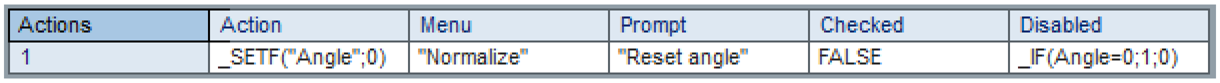
Each user command is specified by a separate command (**Action**). User command (**Action**) defined by a set of parameters in the properties table, see figure below. The first field is **Action** – is actually the action itself, described by the **ConceptDraw** tabular formulas and functions. Most commonly the ***_SETF*** function is used. It establishes a specific property of the object at a certain value. **Menu** - is a name of the command visible to a user. **Prompt** – is a command hint. **Checked** and **Disabled**- are properties that determine the command's marking and running/availability status.





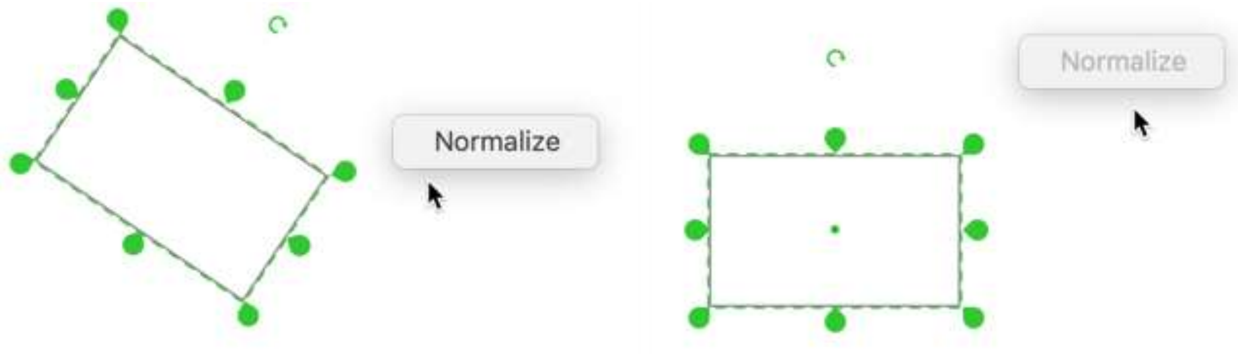
Actions	Action	Menu	Prompt	Checked	Disabled
1	0	""	""	FALSE	FALSE

For example, the reset command for the rotation angle of the object can be defined as follows: according to a user command the object's rotation angle will be reset. The command will be named Normalize. The command is available to perform only if the angle is not zero. Here is an example of this command for the object.



Actions	Action	Menu	Prompt	Checked	Disabled
1	_SETF("Angle";0)	"Normalize"	"Reset angle"	FALSE	_IF(Angle=0;1;0)

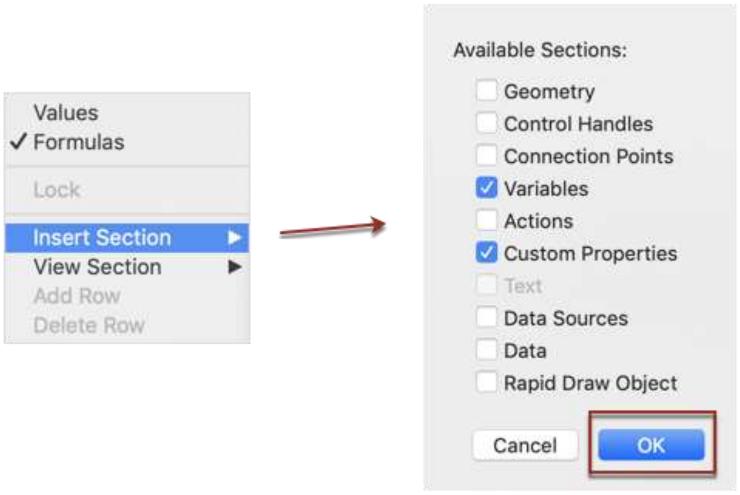
The result is a predefined action, which is available only when the rotation angle is not zero.

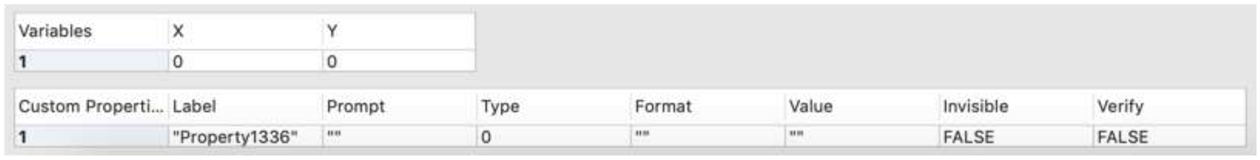


For more complex object of this type, a container to store the internal state of an object can be required. For this purpose you can use the **Variables** and **Custom Properties** that enhance the opportunities of an object. The main difference between these parameters:

Variables can store an integer number, real number or a Boolean value and thus is not accessible to the user. **Custom Properties** container can contain other types of data: a line, color, an element from the predetermined set, **Custom Properties** are available to the user via the interface – Custom Properties dialogue.

Both **Variables** and **Custom Properties** sections are available in the object properties table.





Variables	X	Y
1	0	0

Custom Properti...	Label	Prompt	Type	Format	Value	Invisible	Verify
1	"Property1336"	""	0	""	""	FALSE	FALSE

3.4. Dynamic State Drawing objects

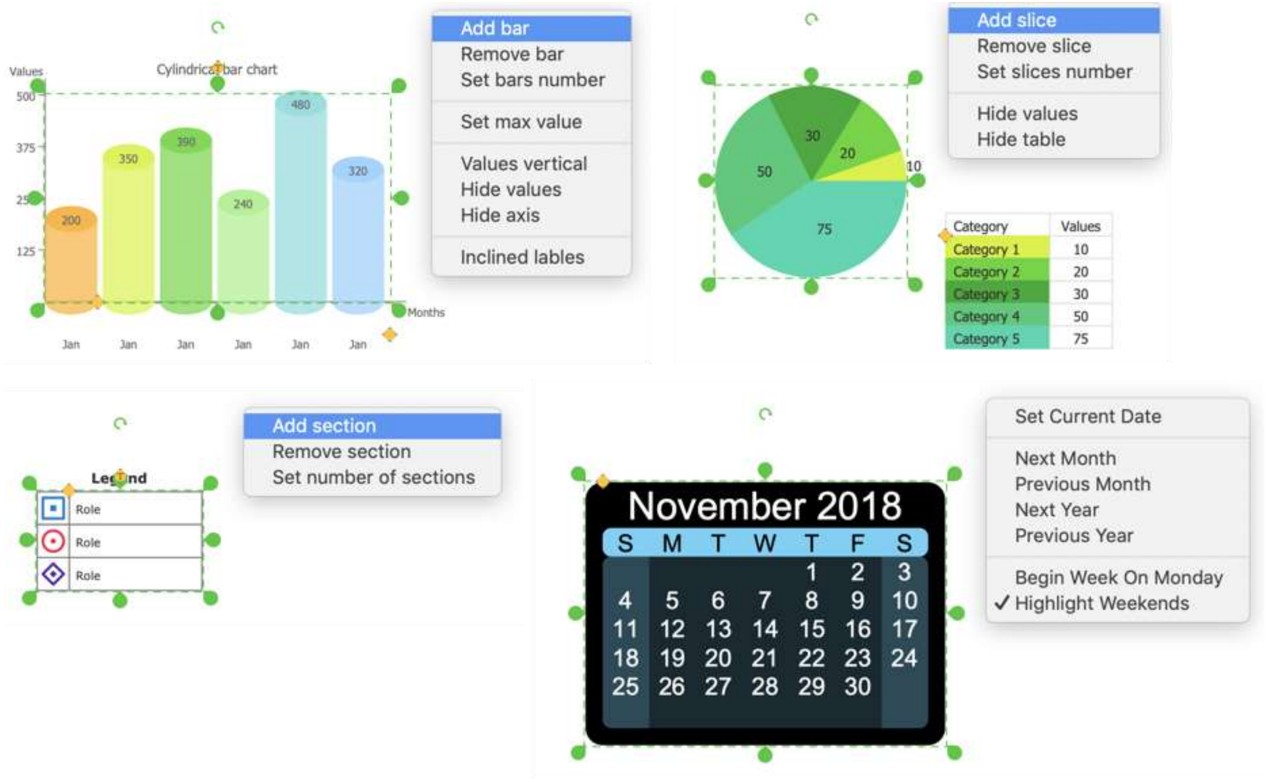
Dynamic State Drawing (Dynamic Variable Objects) - objects that can change its state and type in an unlimited range. As a rule there are different graphics, histograms, pie charts, tables, complex objects, which may have a large number of child objects.

This group includes objects that can have graphic content varies in a wide range, for example an object can have an unlimited number of child objects, columns in the table of sections in the pie chart. Generally there are tables, lists, charts, graphs, pie charts, etc.

Such objects are managed by means of predefined commands (**Actions**) and custom properties (**Custom Properties**). To process the data (variable length) using of a properties table is not enough, so the Dynamic State Drawing objects have **CDBasic Script**, which change their appearance. To create these objects you have to know the basic skills in programming and using **Basic** - like languages.

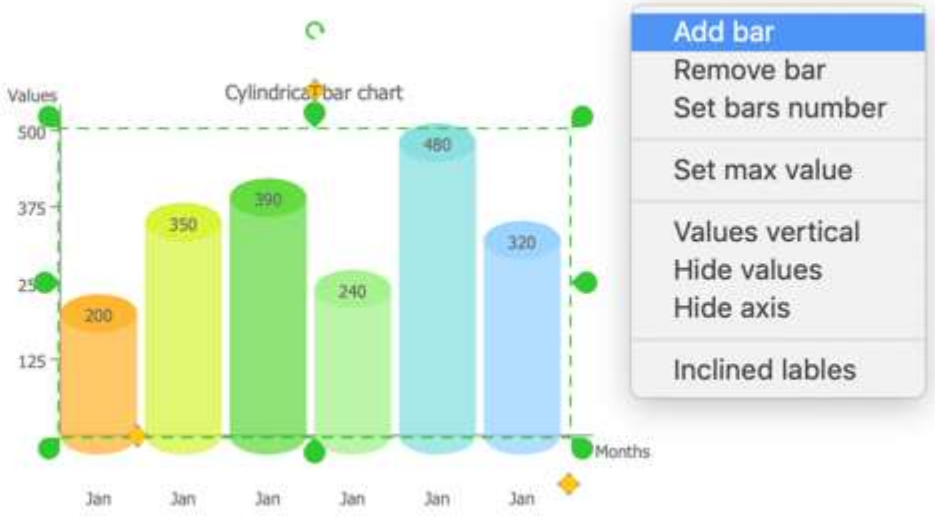
The user data for such objects can be assigned through the **Custom Properties**, a special input field (the InputBox function of CDBasic), or through the texts of other / child objects.

Examples of such objects are shown in Figure below.



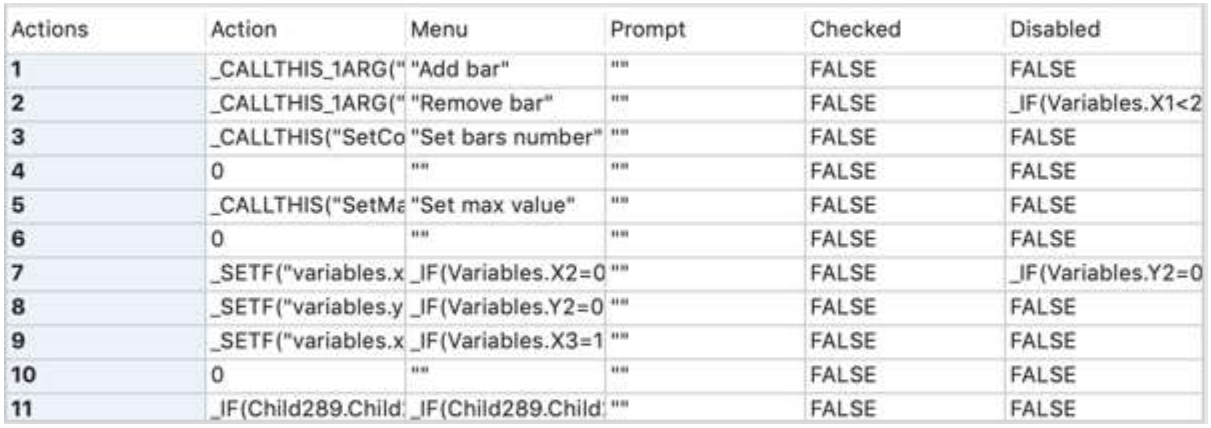
To create such an object, you must first draw it the base image, then assign custom commands (**Actions**), and then connect the call to the appropriate link **CDBasic Script** with call to the custom commands (**Actions**).

Below you can see the histogram object with a list of user commands (**Actions**), also its Actions section of the properties table, and a fragment of the **CDBasic Script**.



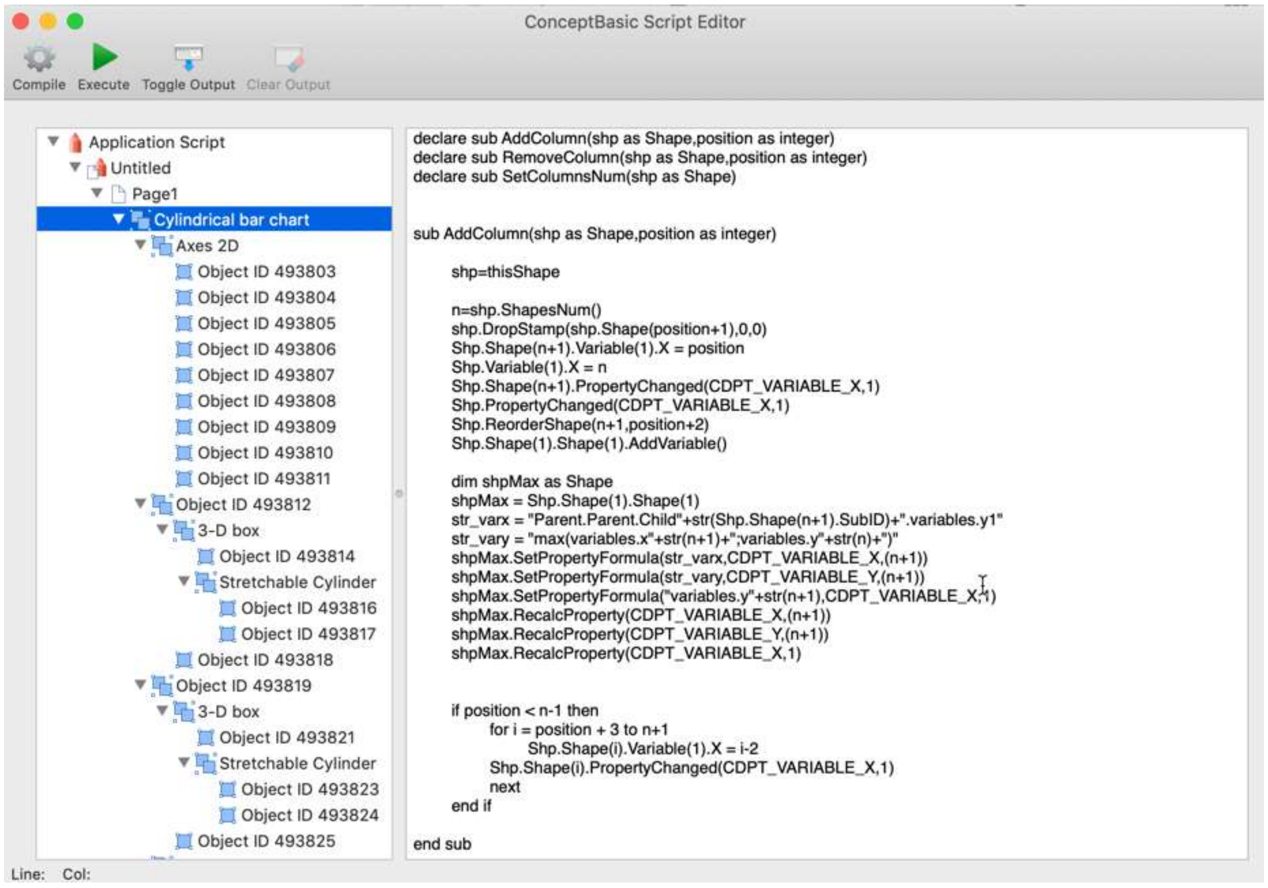
First part of the figure shows that the object has six user commands (**Actions**).

In the second part we can see a fragment of a table with all the properties of these commands.



Actions	Action	Menu	Prompt	Checked	Disabled
1	_CALLTHIS_1ARG("	"Add bar"	""	FALSE	FALSE
2	_CALLTHIS_1ARG("	"Remove bar"	""	FALSE	_IF(Variables.X1<2
3	_CALLTHIS("SetCo	"Set bars number"	""	FALSE	FALSE
4	0	""	""	FALSE	FALSE
5	_CALLTHIS("SetMa	"Set max value"	""	FALSE	FALSE
6	0	""	""	FALSE	FALSE
7	_SETF("variables.x	_IF(Variables.X2=0	""	FALSE	_IF(Variables.Y2=0
8	_SETF("variables.y	_IF(Variables.Y2=0	""	FALSE	FALSE
9	_SETF("variables.x	_IF(Variables.X3=1	""	FALSE	FALSE
10	0	""	""	FALSE	FALSE
11	_IF(Child289.Child	_IF(Child289.Child	""	FALSE	FALSE

In the third part we can see a fragment of **CDBasic Script**, which runs by the user's command **AddColumn**. Objects can contain additional data that can be stored in the **Variables** and **Custom Properties**.



```
declare sub AddColumn(shp as Shape,position as integer)
declare sub RemoveColumn(shp as Shape,position as integer)
declare sub SetColumnsNum(shp as Shape)


sub AddColumn(shp as Shape,position as integer)

    shp=thisShape

    n=shp.ShapesNum()
    shp.DropStamp(shp.Shape(position+1),0,0)
    Shp.Shape(n+1).Variable(1).X = position
    Shp.Variable(1).X = n
    Shp.Shape(n+1).PropertyChanged(CDPT_VARIABLE_X,1)
    Shp.PropertyChanged(CDPT_VARIABLE_X,1)
    Shp.ReorderShape(n+1,position+2)
    Shp.Shape(1).Shape(1).AddVariable()

    dim shpMax as Shape
    shpMax = Shp.Shape(1).Shape(1)
    str_varx = "Parent.Parent.Child"+str(Shp.Shape(n+1).SubID)+".variables.y1"
    str_vary = "max(variables.x"+str(n+1)+";variables.y"+str(n)+")"
    shpMax.SetPropertyFormula(str_varx,CDPT_VARIABLE_X,(n+1))
    shpMax.SetPropertyFormula(str_vary,CDPT_VARIABLE_Y,(n+1))
    shpMax.SetPropertyFormula("variables.y"+str(n+1),CDPT_VARIABLE_X,1)
    shpMax.RecalcProperty(CDPT_VARIABLE_X,(n+1))
    shpMax.RecalcProperty(CDPT_VARIABLE_Y,(n+1))
    shpMax.RecalcProperty(CDPT_VARIABLE_X,1)


    if position < n-1 then
        for i = position + 3 to n+1
            Shp.Shape(i).Variable(1).X = i-2
        Shp.Shape(i).PropertyChanged(CDPT_VARIABLE_X,1)
        next
    end if

end sub
```

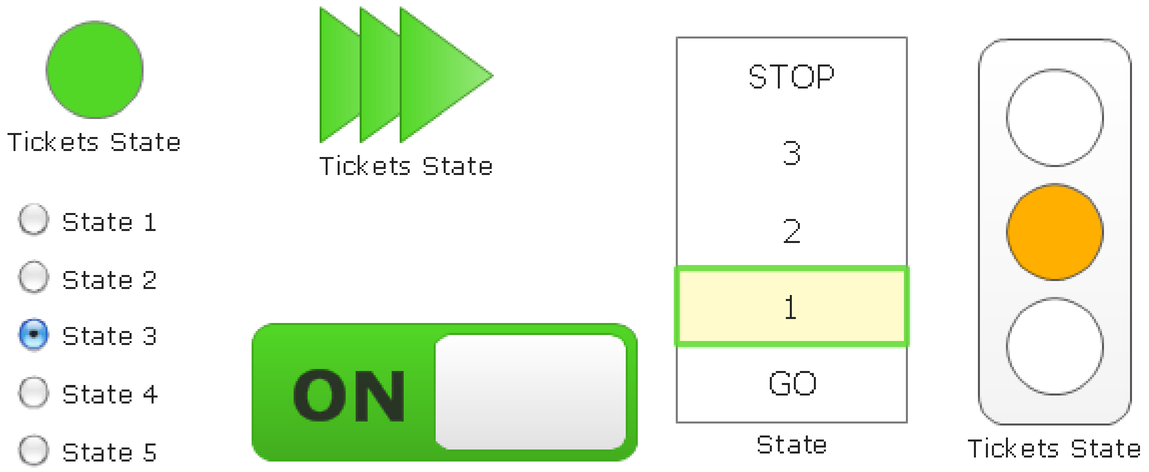
3.5. Live Objects

Live Objects - are the special graphic objects that have a predetermined logic. Form and reaction of the Live Objects depend on the external user data. Live Objects could be very simple, for example – an object displays a single value from a text file, as well as and quite complex – displaying the date from multiple sources. Live Objects used in dynamic presentations and DashBoard systems.

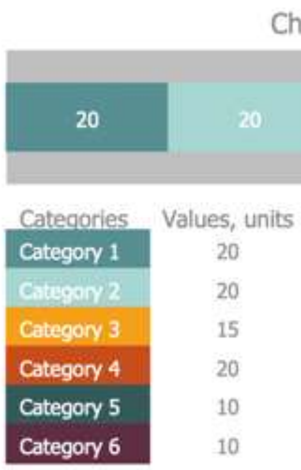
To work with external data, such objects contain the **Data Sources** (a block of properties defining an external file, the data source), through the **Data Sources** data is written to the object properties, commonly to the **Variables**, **Custom Properties**, and then used in the calculations.

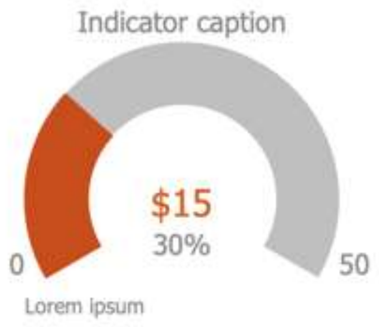
Live Objects could be of two types - the objects of state and dynamically extensible objects. The objects of the first group have a limited set of states, and display data in these states. The second group of objects can be modified in an unlimited range, depending on external data.

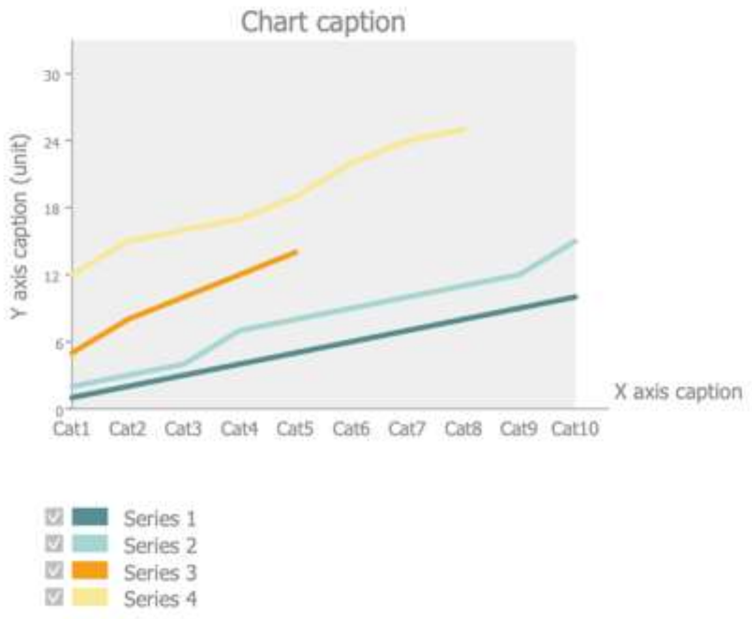
Examples of the live objects of state are shown in Figure.



Examples of the dynamically extensible objects are shown in Figure below.

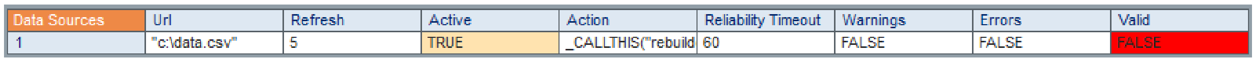






Simple objects Live Objects can be created using the object properties table, to create complex objects it is necessary to implement the certain part on **CDBasic Script**.

A key element for the Live Objects is the data source. Every Live Object has at least one **Data Sources**, which is described by the properties in the table's section.



Data Sources	Url	Refresh	Active	Action	Reliability Timeout	Warnings	Errors	Valid
1	"c:\data.csv"	5	TRUE	_CALLTHIS("rebuild	60	FALSE	FALSE	FALSE

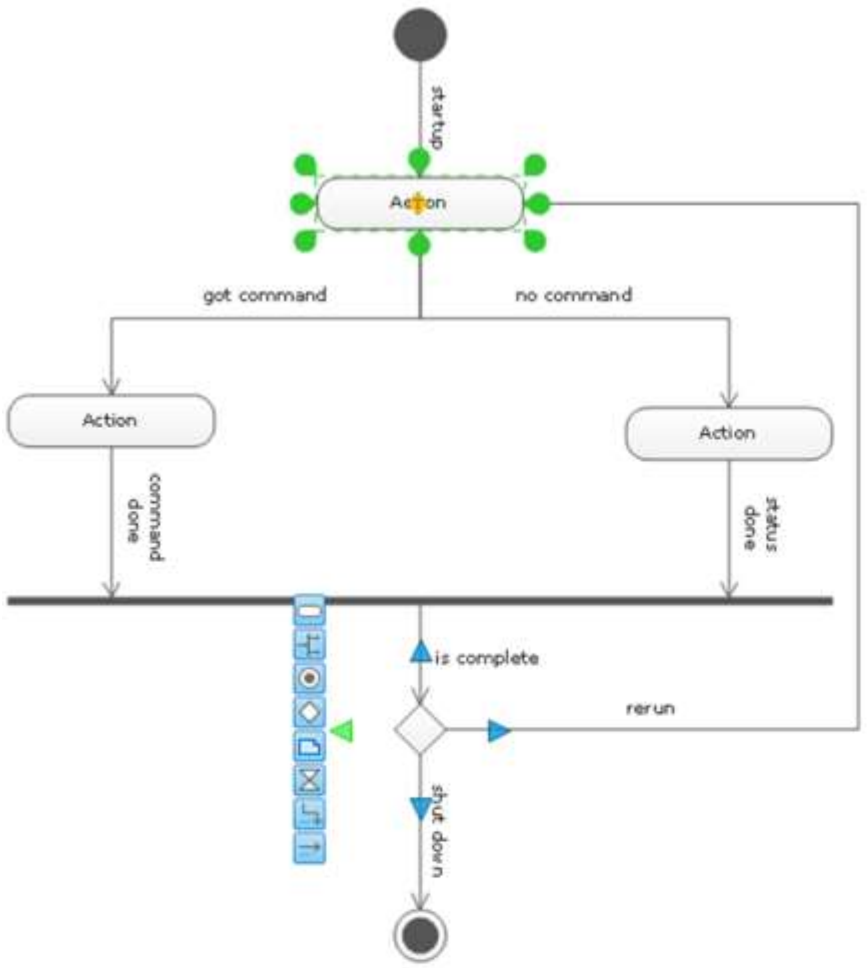
Information, displayed for each data source contains: the path source, frequency of updating, an action that should be done when updating the data (optional).

An object can store / cash the external data and can directly use them from the data source.

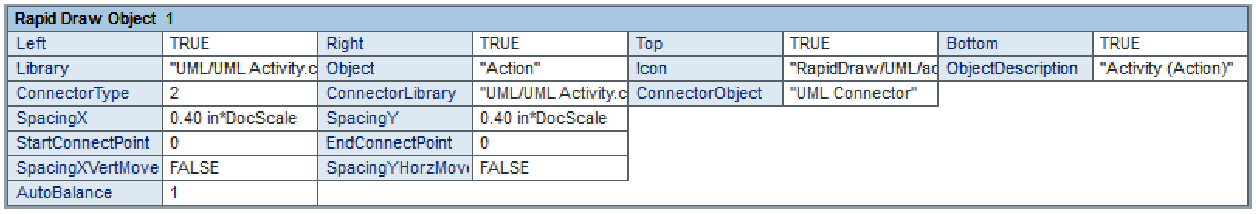
3.6. RapidDraw objects

RapidDraw objects are the objects that implement the mode of rapid creating of block diagrams. **ConceptDraw** has a special mode (**RapidDraw**) using which you can create, combine and arrange new objects on a worksheet automatically. With **RapidDraw** mode help, you can quickly build various block diagrams. Also you can create your own sets of RapidDraw objects.

The following are illustrations of the **RapidDraw** mode.



To create RapidDraw objects you need to create a basic set of objects, keep it in the library, and then activate the **RapidDraw** mode for each object. All configurations are carried out through the property table.



Rapid Draw Object 1							
Left	TRUE	Right	TRUE	Top	TRUE	Bottom	TRUE
Library	"UML/UML Activity.c	Object	"Action"	Icon	"RapidDraw/UML/ac	ObjectDescription	"Activity (Action)"
ConnectorType	2	ConnectorLibrary	"UML/UML Activity.c	ConnectorObject	"UML Connector"		
SpacingX	0.40 in*DocScale	SpacingY	0.40 in*DocScale				
StartConnectPoint	0	EndConnectPoint	0				
SpacingXVertMove	FALSE	SpacingYHorzMov	FALSE				
AutoBalance	1						

Each RapidDraw object is described in a separate section of the table properties:

left-right-top-bottom indicates the sides of the object where the object can be created.

Library displays the path to the library where object is stored.

Object indicates the name of the object.

Icon and **ObjectDescription** contain the supporting information of the object - the icon and the signature.

ConnectorType – indicates the type of connector: 0-smart 1- direct, 2-custom from the library. The latter require specifying exactly the ConnectorLibrary and ConnectorObject.

Spacing - gives you the flexibility to manage the placement of objects by setting arbitrary spacing. This parameter is optional and used in limited cases.

Start-End-ConnectionPoint - make it possible to identify specific points of attachment for connectors. This parameter is used in limited cases.

Autobalance - controls the placement, down-to-right or down, left / right.

3.7. Complex Object

ConceptDraw objects can be of varying complexity. You can group objects and operate with group like with single object, called Group. You can create objects that can with other objects on the page of the document.

ConcetpDraw document has one or more pages. Each page can have a set of graphical objects. Graphical objects can be simple - shapes, and complex - groups. Complex objects, in turn, can contain simple as well as other complex objects.

Any object can use **ConceptDraw** data from other objects. To install the dependencies, one should use the set of the special commands.

3.7.1. The interaction of objects in the table of properties

By using a table of properties, you can set the dependences of properties within object. You can also set the dependences between properties of different objects using *Parent, ObjID* and *Child* constructions.

Parent –provides access to the object-owner of the current object. Is used to establish the dependence of child objects on the variables and values of the parent objects. For example: *Parent.TheText, Parent.GPinX, Parent.Variables.X1.*

ObjID - provides access to the object by the specified identifier. Is used as an *ObjID%_ID%*, where *_ID* is numeric identifier of the object in the document. The *ObjID* construction helps refer to other objects regardless of their level in the hierarchy and his belonging of the current object hierarchy. Link to properties of the object with *ID 17* may be of the form: *ObjID17.The*Text, ObjID17.GPinX*, and* ObjID17.Variables.X1*.* Link to *ObjID* used in complex documents, complex objects, or Dashboard.

Child - provides access to the properties of child objects, the operator is only available for objects that are grouped, and have child objects. Access to child objects is provided by the operator *Child% _num%*, where the *_num-child object* is a number of the child object within the group. *Child* operator is typically used in special objects containing a set of similar child objects such as lists, tables, *CheckBox* and *RadioButton* sets. Example of links by *Child: Child17.TheText, Child17.GPinX, Child17.Variables.X1*.

You can combine links *Child* and *Parent*, if you want to get access to top-level object or object at the same level, for example:

- Parent.Parent.Variables.Y2 - get the value of Y2 for parent of parent.

- *Parent.Child3.Gpinx* – get the coordinate on the X axis of a child object from the current parent.

3.7.2. The interaction of objects with the help of ConceptDraw Basic

You can create dependencies of objects' properties with the help of ConceptDraw Basic.

As mentioned earlier, all objects are placed on the page of the document. To get a pages list, the document has the following methods:

PagesNum – returns the number of pages in the document

PageByID - returns a page by its ID

Page - returns a page at the specified index.

Each page contains a number of methods for working with a set of objects located on it:

ShapesNum - returns the number of objects on the page

ShapeByID - returns a graphics object by its identifier

Shape - returns a graphics object by its index.

Each graphical object has a special method of *Parent* that returns the parent graphical object (if exist), or the Page where is the graphic object placed.

The grouped objects have the same set of methods for child objects processing, like page, namely:

ShapesNum - returns the number of objects on the page

ShapeByID - returns a graphics object by its identifier

Shape - returns a graphics object by its index.

Thus, to enumerate all the objects on the page and replace the text object with the "unknown" to "newtext", Basic Script code will look like:

```
Dim shp as shape
```

```
For i=0 to thispage.shapesnum
        Shp = thispage.shape(i)
        If shp.text=”unknown” then
                shp.text=”new text”
        End if
Next if
```

4. Task Solving.

4.1. Classification of problems solved by the developer.

Using the arsenal of developer tools, one can solve different tasks on user data visualization, automatic provision of information and graphical reports generating.

There are five types of visualization tasks to be solved with the help of **ConceptDraw DIAGRAM**:

1. Custom templates (custom templates). The task involves the creation of a specific working environment and created templates for the documents. The task is to create a custom template with settings of predefined objects libraries.
2. Smart Objects (Interactive objects). This group includes tasks to create special graphics objects with interactive behavior.
3. Live Objects (live objects, lights). Creating the special graphic objects changing its state by changing the external data.
4. RapidDraw template (templates for creating charts with the help of technology RapidDraw). Creating a custom set of graphic objects to quickly create charts for specific topics, such as flowcharts, ERD, BPMN.
5. Visual Reports (visual graphic reports). Creating a visual report on the results of the action of any system. The problem reduces to the generation of CD XML file describing the structure of a graphical report.

4.2. Examples of solved tasks.

Below is a common scenario for each of the types of tasks.

4.2.1. Custom Templates

Custom templates are a blank for document. The template has the entire set of properties Custom templates provide a blank document. The template has the entire set of properties inherent in the document: title, page size, units, grid settings, snapping and other.

A template can have a set of pages, and graphics. When you open a template opens the associated library of graphic objects.

To create a custom template, make the following steps:

1. Create a set of graphical objects used in the template
2. Create a new library, moved here the graphic objects and save it.
3. Create a new document (template), set it to the required settings, place graphic objects on it
4. Save the document as a template, with *. cdt expansion.
5. While opening the saved template, the corresponding libraries open automatically and it will be ready for new document creation.

4.2.2. Smart Objects

Smart Objects can be created only by using the table properties. Also you may need to use **ConceptDraw** Basic.

Behavior of the Smart Objects usually defined by custom **Actions** and **Control Handles**.

The general procedure of the Smart Objects creation is the following:

1. Create a simple graphic image using simple graphic tools and ready library objects.
2. Add Custom **Control Handles** using the object table of properties of.
3. Set the depending of the object properties on the position of **Control Handles** (set formula), test the object performance.
4. Add custom commands (**Actions**) for objects
5. Set the action by the commands - change the properties, call script **CDBasic**.
6. Write and debug the script, if needed.
7. Check the object performance.

This category includes State Drawing and Dynamic State Drawing objects.

4.2.3. Live Objects

This category of objects makes it possible to display data from the files of external URL.

Typically, users use the ready Live Objects, and they need only to set the data source object, then the object will read the data and change its appearance.

The creation of the Live Object requires knowledge of object properties table. To create a Live Object you must have a sample data source file.

The procedure to create a Live Object is following:

1. Draw a simple graphic object, possibly with sub-objects, which will change its appearance by the external data changes.
2. Add the **Data Source** property - Data Source. For this purpose, call the Insert Section - Data Source command in the properties table.
3. Set the data source properties - the path to the file of source, the frequency of data updates. If the data source file is correct the "**Valid**" property will get the TRUE value.
4. Establish the dependence of the object properties from external file data. The function **CSV Value**, returning the cell value from a CSV file is typically used for this purpose.
5. If you need to calculate intermediate data, you can add user-defined variables and properties - the **Variables** and **Custom Properties** sections of the properties table.
6. It is recommended to use the long periods of data sources updates - from 5 seconds to a minute.
7. You can set the dependence of the properties of one object from several different data sources. Just add one or more data sources to the object.

4.2.4. RapidDraw objects

RapidDraw objects allow the user to quickly build a block diagram from a single, simple object. Technique uses the drawing tools right on the workspace.

To create your own libraries of RapidDraw objects you need just basic knowledge on the work with the object properties table. A typical scenario for creating such objects is following:

1. Create a set of simple graphical objects to be used in the RapidDraw drawing mode.
2. Create a new library. Save all necessary objects into it. Save the library.
3. For the first object define a set of other objects to build. For this purpose, use the command Section – RapidDraw in the properties table. Set the necessary properties in the section that appears: the path to the library, the name of the object, the type of connector, the shit of the axes (if needed), and others.
4. Repeat p. 3 for all graphic objects that you want to build from current.
5. Desirable. For every picture draw an icon size of 19 * 19px, and assign an icon for the object in section RapidDraw.

4.3. Installing add-ons, developed, to the user's computer.

There are two main locations of files and folders of ConceptDraw:

Program files location:
«%ProgramFiles%\ConceptDraw Office\ConceptDraw DIAGRAM\» (win)
«/Applications/ConceptDraw DIAGRAM.app/» (mac).

Here the files shared for all users of current computer are located. This is the path to save all the files needed to run the application.

Add-ons, templates, samples libraries and temporary files are located here:
« %ProgramData% \ConceptDraw Solution Park» (win)
«/Library/ConceptDraw Solution Park » (mac).

Typically this path is used during the installation of user defined solutions. Solutions extend the value of ConceptDraw DIAGRAM for professional users. Each user on the current computer can have the own set of solutions.

Shared files:

Path	**Description**
Data\	System Libraries needed to for diagramming and import from other applications
Data\Visual Reports\	Libraries, templates, scripts needed for creation visual reports from ConceptDraw PRJECT.
Dicts\	Dictionaries for spell checker
Help\	Reference on application, CDBasic and CDX file structure
Help\ContextHelp\	Files of the Context help

HTML_Templates\	Templates for export to HTML format
Libraries\	CDL and CDLX libraries. Nowadays contains the DrawingShapes.cdl only. All other moved to the second section.
RapidDraw\	The Rapid Draw objects' button icons
Resources\	Dynamic software libraries with support of localization and languages XML files
Shapes Gallery\	Libraries and Template Gallery icons
Textures\	Image (texture) intended to fill the ConceptDraw DIAGRAM objects

User Files:

Path	**Description**
Backup\	Location of documents Autosave
Index\	Data for quick search through documents and libraries.
Libraries\	CDL and CDLX Libraries
RapidDraw\	Icons for RapidDraw objects of the solutions, installed.
Samples\	Document samples
Templates\	Document templates

The developer of solution must create the sub-folders for his files in the following second section folders: Libraries, RapidDraw, Samples and Templates. The sub-folder should be named the same as solution.

When the application runs the necessary templates and examples will be displayed in a separate category in the Templates Gallery. In the folder **Templates \ <solutionname> \ Localize** should be placed the files of solution localized descriptions, such as English language - file DescriptionEn.html.

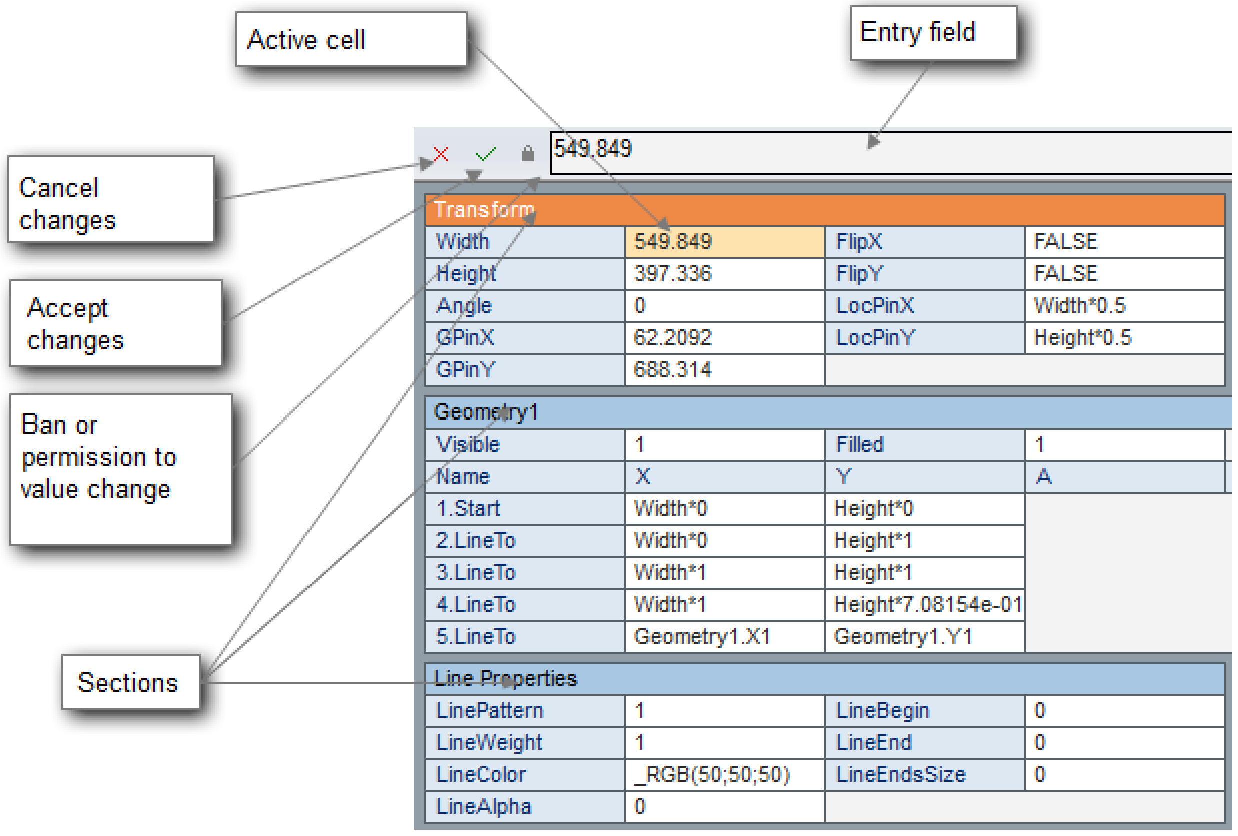
5. Data Management

5.1. Managing the object's data via table

Each object in the **ConceptDraw DIAGRAM** is built on the basis of dozens of parameters. They can be numeric, string or formulas. All parameters are collected in the **object parameters table**, where each of them can be changed, thereby altering the properties and behavior of the object. Managing the object via table's means creating in the table of properties the relationships between various parameters of the current object and other objects using formulas.

5.1.1. Ways to work with the table

To access the table of parameters, select object and then press **F3**, or **PowerEdit** button in the **Properties** section of the **Shape** ribbon tab. Table will open in a separate window.

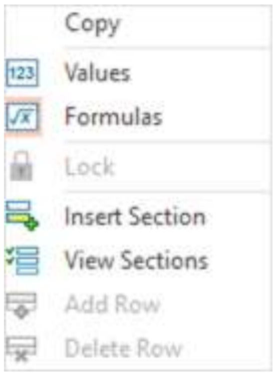


549.849

Transform			
Width	549.849	FlipX	FALSE
Height	397.336	FlipY	FALSE
Angle	0	LocPinX	Width*0.5
GPinX	62.2092	LocPinY	Height*0.5
GPinY	688.314		

Geometry1			
Visible	1	Filled	1
Name	X	Y	A
1.Start	Width*0	Height*0	
2.LineTo	Width*0	Height*1	
3.LineTo	Width*1	Height*1	
4.LineTo	Width*1	Height*7.08154e-01	
5.LineTo	Geometry1.X1	Geometry1.Y1	

Line Properties			
LinePattern	1	LineBegin	0
LineWeight	1	LineEnd	0
LineColor	_RGB(50;50;50)	LineEndsSize	0
LineAlpha	0		



It is possible to use the following table elements:

• In the **input field** you can edit the contents of the selected cell (enter a value or formula);

• Buttons can deny or allow, accept, discard the changes after editing the selected cell;

• **Sections** join the groups of parameters: the size, properties, geometry, lines, fills, text, behavior parameters and other;

• Cells contain a value or a formula for each object parameter. You can select a cell and change its content using the input field;

• **Menu** contains the commands that can be applied to the table (copy the data, change the data displaying mode, discard changes, edit the section, etc.).

There is an **input field** at the top of the window. If you select any cell in the table, its contents will appear in the input field editable. Double-clicking on the cell, in addition, will select an editable string. To finish editing you must press **Enter**, to roll back - **Esc** (or use the icons on the left of the input field). If you make a mistake, while editing sell, for example, input letters instead of numbers, or enter the illegal symbols, the program will display an error message and will roll back to its original condition.

Cell is a place to store the object parameter. The active cell is available for editing, which occurs in the input box above the table. To quickly start editing, you need to double click on the cell. To change the active cell, you can use the arrow keys.

The content of a cell depends on the mode of viewing of the table. The **Values** and **Formulas** modes are available. The switching between modes is possible via the context menu. If the **Values** mode is ON, the cells will display values in the current measurement units. If the **Formulas** mode is ON the cells will display the formulas. If the cell does not contain the formula, the values of length in tenths of a millimeter, or angles - in radians will display.

Note that the edit field in any mode, displays the formula (if exist). You can enter data in units other than those specified in the document. For example: 12 in, 3.5 ft, 0.66 m. While editing the content of a cell you can use data from another cell by clicking on it with the mouse:

Single mouse click on the other cell	Insert formula from this cell into the input field
/Ctrl + click on the other cell /Cmd + click on the other cell	Insert the number value of the cell into the input field
/Alt + click on the other cell /Opt + click on the other cell	Transfer the names of the cell (to refer to it)

5.1.1.2. Operators

In the table formulas you can use arithmetic operators, comparison operators and logical operators.

Arithmetic operators

• Exponentiation operator "^" or "**"

• Change of the sign and subtraction "-"

• Addition "+",

• Multiplication and division - "*", "/"

• Integer division "\",

• An arithmetic modulus (remainder of the division) «MOD»,

• Connection string "&".

Comparison operators

• Less than "<",

• More ">"

• Less than or equal to "<="

• Greater than or equal to "> =",

• Equal "="

• Not equal to '<>'.

Logical operators

Logical operators can be used in formulas except the logic functions *_AND(), _IF(), _NOT(), _OR()* and *_XOR()*.

• Conjunctions «AND»,

• Disjunction «OR»,

• Exclusionary "or" «XOR»,

• Denial «NOT»,

• Equivalence «EQV @?

• Implication «IMP».

5.1.1.1. Formulas

Write a formula of object parameter for setting the object to associate it with the other parameters, or user actions.

*Variables.X1+(Variables.X2-Variables.X1)*0.293*

_MIN(0;Geometry1.X2-Variables.X2)

=_IF(Variables.X1=1;_SETF("Variables.X1";0);_SETF("Variables.X1";1))

To refer to the contents of the cell in the formula, you must specify its name (it displays by blue color near the cell), for example: Width, TextAngle, etc.

<cell name>

For the sections where you can add cell (Sections Geometry, Controls, Connect, Variables, FontFormat, ParagraphFormat, Actions) a different system of treatment is used:

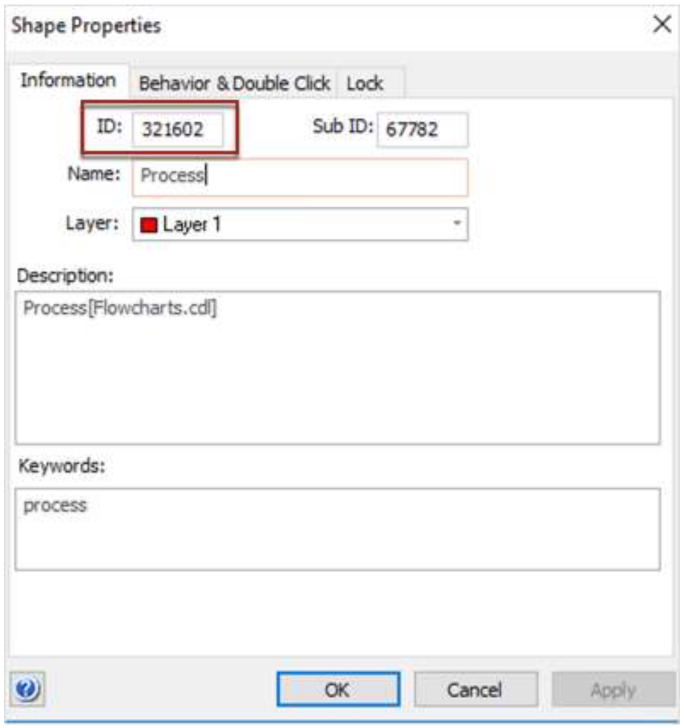
<partition name>. <cell name>

The **name of the cell** is formed from the name of the column and row sequence number. For example:

"Controls.YDyn1"; "Variables.Y2"; "Connect.X1".

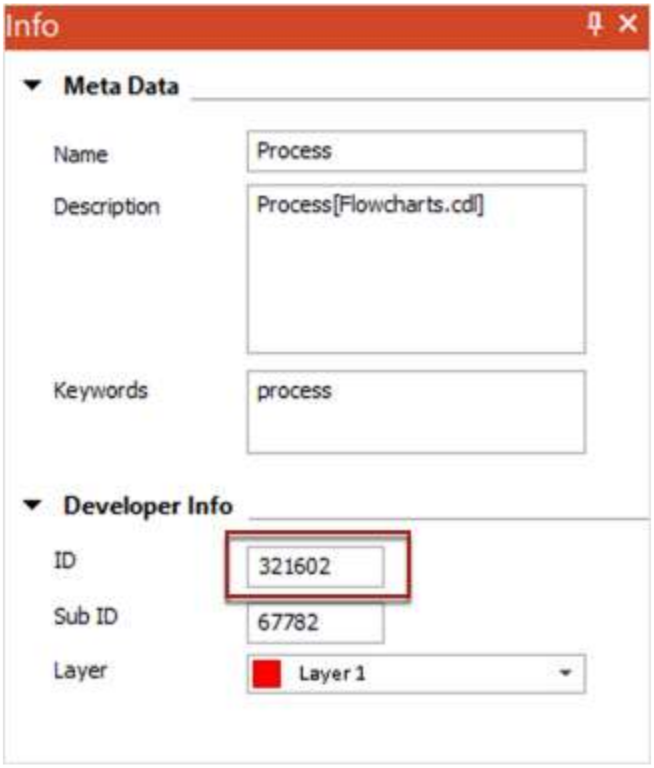
In the **Geometry** section, the cell name also includes the number of the section (as the geometries may be several). For example: "*Geometry1.Y1*", "*Geometry2.C2*". For the first two cells of the names look like this:"*Geometry1.Visible*" and "*Geometry1.Filled*".

You may need to reference the cells, describing other objects. Each object has an identification number - it can be seen in the header of the table window or in the **Information** tab of the **Shape Properties** dialog.



ID: 321602
Sub ID: 67782
Name: Process
Layer: Layer 1
Description:
Process[Flowcharts.cdl]
Keywords:
process

Or in the **Info** panel (parameter **ID**)



Name: Process
Description: Process[Flowcharts.cdl]
Keywords: process
ID: 321602
Sub ID: 67782
Layer: Layer 1

This ID is used to refer the object’s parameters

<object ID >.<Section name>.<Cell name>

For example:

ObjID13.Geometry2.X1

ObjID2.Width

If the object belongs to a group, then to access the parameters use the prefix **Parent**.

Parent.<section name>.<cell name>

For example:

Parent.Height

The reference to objects - members of a group should be written as:

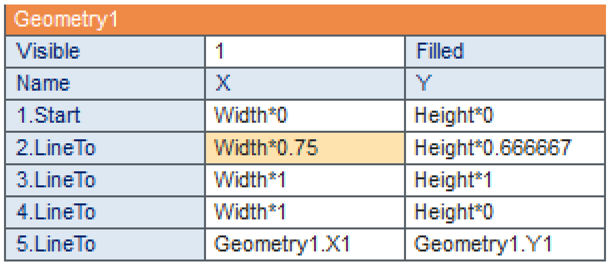
Child<number>.< section name >.< cell name >

For example:

Child2.Angle

The number in the name of the object corresponds to its order number in the group. This number can be found out in the **Information** tab of the **ShapeProperties** dialog or in a floating dialog **Information** (Field **SubID**).

Some formulas will automatically appear. They are called **formulas by default**. For example, if you create a line, one of its parts is prescribed as follows:



Geometry1		
Visible	1	Filled
Name	X	Y
1.Start	Width*0	Height*0
2.LineTo	Width*0.75	Height*0.666667
3.LineTo	Width*1	Height*1
4.LineTo	Width*1	Height*0
5.LineTo	Geometry1.X1	Geometry1.Y1

The expressions *"Width * 0.75"* and *"Height * 0.666667"* are the formulas by default. Due to the formulas, when the object resized, its vertexes saves their positions relatively to the control frame. Note, that formulas by default will be converted when moving the vertex by mouse (and will be ready for a new changes).

If there is a “=” in front of the formula, it means that the formula cannot be changed by any action, but direct cell editing. If there is no defense, it could become a default one, for example, when you move the control point.

If the parameters are used to indicate linear dimensions (width, height) or coordinates, the measurement units should be listed. For example, the parameter **Width** of the **Transform** section:

=15 mm

If the unit is not specified, a tenth of a millimeter (0.1 mm) will be used as default measurement **unit**.

The default measurement units for angles are radians. If you want to record the angle in degrees, you need to write the word "deg".

= 30 deg

5.1.1.3. Functions Call

In addition to simple expressions from the variables and operators in the formulas you can use the built-in table functions.

All table functions begin with an underscore, and are written in caps letters. For example: *_CALLTHIS(), _MIN(), _LN()* (see Appendix 1..). But you can type the functions without these symbols. The editor will automatically convert the characters to meet these requirements.

The parameters passed to the function are indicated in parentheses, next to the function name. Function can have several parameters, or even does not have them at all. If the function has several parameters, they are listed, separated by semicolons. The transmitted parameters may be of various types - string, integer, floating-point value, Boolean, etc. The parameters and their types required for certain function is described by Appendix 1 The Functions and operators of object properties table.

The syntax of the functions on the example of a function *_IF()* (Appendix 1.)

_IF(arg1;arg2;arg3)

If the value of *arg1* is- a non-zero number or non-empty string, the function returns *arg2*, otherwise - *arg3*.

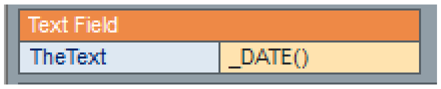
Examples:

_IF (2> 1, 3, 4) - the function returns a value of 3

_IF (""; 3, 4) - the function returns a value of 4

The table functions call on an example of a function call *_DATE ()* (see Appendix 1.):

For example, you want the object **ConceptDraw DIAGRAM** displaying the current date. To do this, draw a rectangle, add a table in his section of the **TextField** (from the table using the Insert Section dialog, or just writing something in it) and enter the name of the function **TheText** *_DATE ().* So you will call *_DATE (),* which will return a string with the current date and this will be the value of a cell line **TheText** object.



Text Field	
TheText	_DATE()

The task is solved.

5.1.1.4. Processing of the object events using formulas

ConceptDraw DIAGRAM can process some events, such as changing the position of the object on the page; change its size and the angle.

The program allows you to use the event to set the behavior of objects in the document by specifying in what cases the parameters of the object should be updated.

The events are specified in the formulas, after the semicolon.

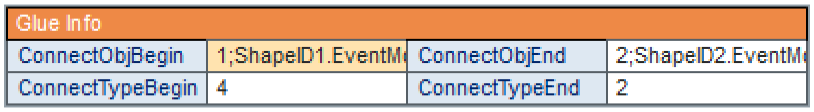
<formula content>; <event1>; < event 2>; < event 3>

When an event occurs, the formula is updated, and also updates the content of other fields from a table of parameters, that refer to this formula.

Example:

You can see the event processing on the example of connector, which is connected to the object in the two points.

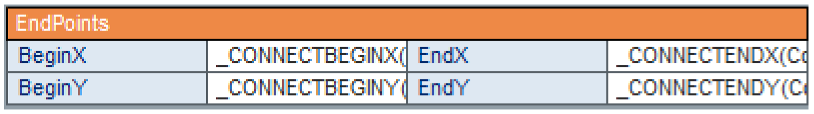
The fields of the table Glue Info;



Glue Info			
ConnectObjBegin	1;ShapeID1.EventM	ConnectObjEnd	2;ShapeID2.EventM
ConnectTypeBegin	4	ConnectTypeEnd	2

ConnectObjBegin	1;ShapeID1.EventMove
ConnectObjEnd	2;ShapeID2.EventMove

The event, which initiates updating of parameters (in this example it is moving of the object) is specified after the semicolon. Note that the objects ID themselves does a not change, but they affect the parameters from the EndPoints section:



EndPoints			
BeginX	_CONNECTBEGINX(	EndX	_CONNECTENDX(C
BeginY	_CONNECTBEGINY(	EndY	_CONNECTENDY(C

BeginX	_CONNECTBEGINX(ConnectObjBegin;ConnectObjEnd;ConnectTypeBegin)

BeginY	_CONNECTBEGINY(ConnectObjBegin;ConnectObjEnd;ConnectTypeBegin)
EndX	_CONNECTENDX(ConnectObjBegin;ConnectObjEnd;ConnectTypeEnd)
EndY	_CONNECTENDY(ConnectObjBegin;ConnectObjEnd;ConnectTypeEnd)

If the same event (Event Move) is specified for each parameter in the EndPoints section, there are no needs to specify it in the Glue Info section.

Object events.

There are several events that can be specified for each object. They are specified in the formula as follows:

[<object name>.] <event name>

Name	**Responsible for**
EventMove	The movement of an object
EventResize	The resizing of an object
TextEventMove	The movement of an object’s text frame
TextEventResize	The resizing of an object’s text frame
CharPropEvent	The changing of the Character Properties section of the parameters table and the object’s text format

Changing of any variable in the table of parameters, also, may be an event. For example, if one writes **<object name>. Width** after the semicolon in a formula, then the update of the formula and update of the formulas that refer to this formula will only occur when the length of the specified object is changed.

5.1.1.5. Variables contain information about the document

In addition to the variables from the object parameters, the variables specified in the **Properties** dialog that support the document properties, , also can be used in a formula:

DocTitle	Title of the document
DocSubj	Purpose of the document
DocAuthor	Author of the document
DocCompany	Organization

DocDesc	Description of the document
DocSnapSens	Sensitivity of sticking to the grid - in pixels
DocPageSizeX	Returns the width of the page in the specified units
DocPageSizeY	Returns the height of the page in the specified units
DocShadowOffsetX	The shadow shifting to the right
DocShadowOffsetY	The shadow shifting below
DocScale	Scale of the document
PageName	PageNname (may be changed though the Pages panel by double-click on the page name)

For more information about the **Properties** dialog see the Help articles: Dialogs - Modal dialogs – Properties. For more information about the Pages panel see the Help articles: Dialogues - Floating Panels - Pages.

5.1.1.6. Call to CD Basic subprogram from the table field

There tree table functions to call CD Basic script from the table: *_CALLTHIS(), _CALLTHIS_1ARG() and _CALLTHIS_2ARGS()* (Appendix 1.). The first one can call a CDBasic procedure or function on its name. And the other two allow passing additional one or two arguments.

The called function can be defined at any level, but must have the following view:

Function proc_name (shp As Shape [, Arg1 As <Type1>[, Arg2 As <Type2>]])[As <Type3>]

The variable shp here represents the object from which the function is called. Arg1 and Arg2 are the custom arguments that are passed to the function when called from *_CALLTHIS_1ARG() and _CALLTHIS_2ARGS().*

Notice. Inside the function called from the table, should not be used a couple of methods StartRebuild () - EndRebuild (). To update the table cells the similar commands are launched by the application automatically. Custom calling EndRebuild () can be confusing and cause the breaking of the parameters update process.

For example, let's create a rectangle object with a custom menu item on which the application will ask the new text of the object. At the level of the document we define a function *MyProc()*:

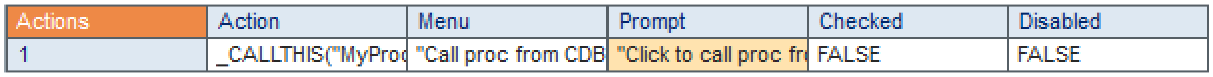
```
Function MyProc(shpAsShape) As Byte

Dim ss As String

ss = InputBox$("Entertextforshape")

shp.Text = ss
```

```
End Function
```

Let’s add the user context menu to rectangle. In the Action field to we write a function call:

```
_CALLTHIS("MyProc")
```

In the field Menu - we write a function name "Call proc from CDBasic script", in the field Prompt - "Click to call proc from CDBasic Script".



Actions	Action	Menu	Prompt	Checked	Disabled
1	_CALLTHIS("MyProc	"Call proc from CDB	"Click to call proc fr	FALSE	FALSE

The rectangle will obtain be a new context menu item "Call proc from CDBasic script", clicking on which you can call the dialog with the proposition to change the text.

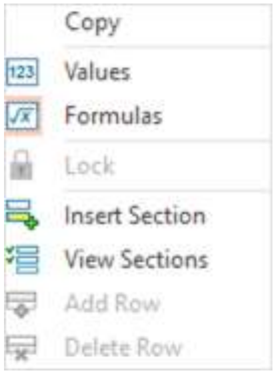
5.1.2. The Table Ssections

The table of object parameters has several sections. Each section is responsible for the certain functionality of an object (such as position, size, text, etc.).

Some sections may be missing because they correspond to an optional feature of the object (for example, control points, or a custom context menu).

You can hide a section, or make it visible. Use the dialogue **View Sections**, which can be opened from the table context menu. Also it is possible to add optional sections, using the dialogue **Insert Section** (menu **Insert Section**).

Some sections (Geometry, Controls, Variables, Character Format, Paragraph Format, Actions) integrate the cells into solid information line of information (for example, describes a control point, line segment shapes, etc.). In such case, one can operate with the whole line, using commands: **Add Row**, **Delete Row**. These commands are available from the context menu of the table.



You cannot delete the entire section. But the optional section will automatically retire, when you remove all of its lines.

One section is active (orange highlight of the section title). It is the section where you can make edits of table cells . To activate the section, you need to click on its title or one of its cells.

Each section can be folded into single line. To do this, make it active, and then click on the title. Second click will open the section.

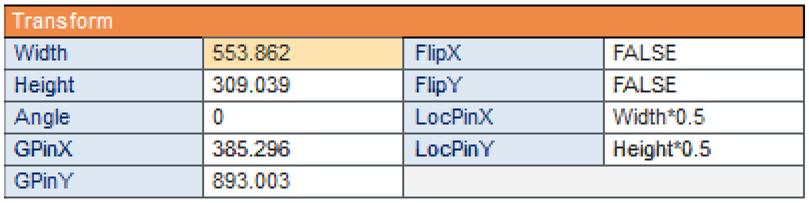
A table can contain up to 22 types of sections. The section of each type (except Geometry and RapidDraw) occurs once or not occurs at all. The Geometry section as well as RapidDraw may attend more than once.

List of sections of the object parameter table:

- Transform;
- Geometry;
- EndPoints;
- Glue Info;
- Variables;
- Controls;
- Connect;
- Actions;
- Data Sources;
- Data;
- RapidDraw Object;
- Custom Properties;
- Line Properties;
- Fill Format;
- Text Field;
- Text Transform;
- Character Format;
- Paragraph Format;
- Text Block Format;
- Text Tabs Table;
- Protection;
- Miscellaneous;

5.1.2.1. Transform

This section contains parameters that are responsible for the location, size and orientation of the object.



Transform			
Width	553.862	FlipX	FALSE
Height	309.039	FlipY	FALSE
Angle	0	LocPinX	Width*0.5
GPinX	385.296	LocPinY	Height*0.5
GPinY	893.003		

The internal coordinate system of the object based on the control frame: the center of the system (reference point) is located in the upper left corner. The axis directed to the right / down.

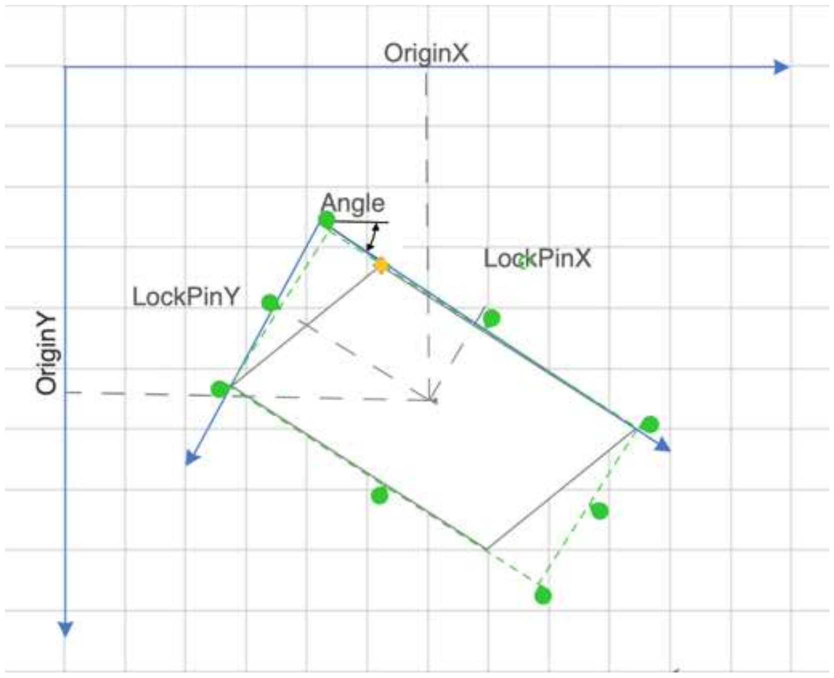
Externally, this system of coordinates attached to the parent group by the following parameters:

• **GPinX, GPinY** define the center of rotation of the object in the external coordinate system

• **LocPinX, LocPinY** - the same point in the internal coordinate system

• **Angle** - the angle of rotation of the internal system relatively to the external

• **FlipX, FlipY** – switching of the mirror image horizontally and vertically

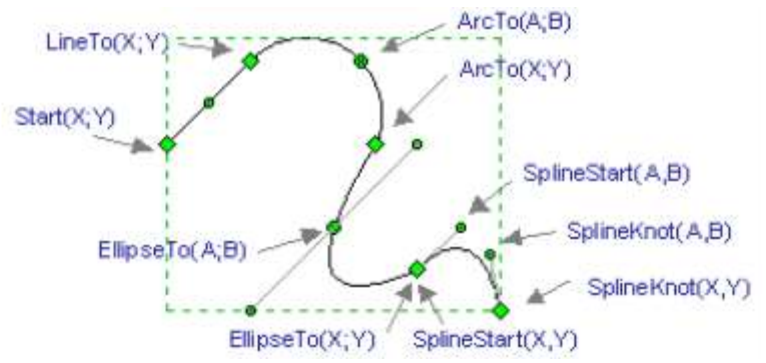


Note: If the object does not have a higher group, it is considered that it is the part of the group, which forms the entire page.

5.1.2.2. Geometry

This section describes the construction of a continuous chain of segments. The object may consist of several chains, so the Geometry sections can also be a few. All of them are numbered, for example: **Geometry1**, **Geometry2**, etc.

The segment can be a point, segment, and arc of a circle or ellipse, spline section.



The segment is based on two points. Arc of a circle - on three:

- the initial
- final
- intermediate point on the arc

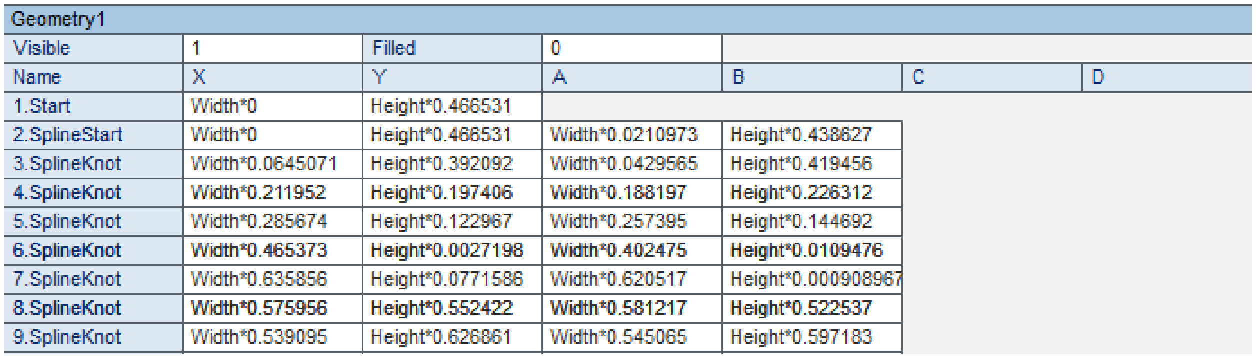
The arc of the ellipse is based on three points and two additional parameters:

- starting point
- end point
- arbitrary point on the circle
- the ratio of the lengths of major and minor axes of the ellipse
- the angle of the semi major axis in the internal coordinates of the object

The segment of the spline is based on four points:

- initial
- initial guidance
- final
- the ultimate guidance

Each segment bases its construction on data of the previous segment. The end point of the previous segment is considered the initial to the current. For the spline segment, in addition, the data on the primary guidance is taken from the previous segment (the ultimate guidance of the previous segment).



Geometry1						
Visible	1	Filled	0			
Name	X	Y	A	B	C	D
1.Start	Width*0	Height*0.466531				
2.SplineStart	Width*0	Height*0.466531	Width*0.0210973	Height*0.438627		
3.SplineKnot	Width*0.0645071	Height*0.392092	Width*0.0429565	Height*0.419456		
4.SplineKnot	Width*0.211952	Height*0.197406	Width*0.188197	Height*0.226312		
5.SplineKnot	Width*0.285674	Height*0.122967	Width*0.257395	Height*0.144692		
6.SplineKnot	Width*0.465373	Height*0.0027198	Width*0.402475	Height*0.0109476		
7.SplineKnot	Width*0.635856	Height*0.0771586	Width*0.620517	Height*0.000908967		
8.SplineKnot	Width*0.575956	Height*0.552422	Width*0.581217	Height*0.522537		
9.SplineKnot	Width*0.539095	Height*0.626861	Width*0.545065	Height*0.597183		

All coordinates are entered in the internal coordinates of the object.

You can specify two additional parameters of the whole geometry in the same section.

- Visible –regulates the visibility of the chain segments. With this flag, depending on the state of an object, you can disable the display of certain geometries. So, for example, realized the effect of adding / removing elements of the object using the context menu.
- Filled – regulates the displaying of the object's fill.

To accurately specify the coordinates of the object within a group in a coordinate system of group or page, you need to know how to convert the coordinates of any object's point from the local system to the external frame of reference. We know the coordinates of rotation center of the object in both systems and we know the coordinates of the object's point in the local coordinate system. We can find the coordinates of the object in the external system by applying the following transformations:

The coordinates of the point – GDotX:

*GDotX = GPinX –[LocPinX*COS(Angle) - LocPinY*SIN(Angle)] +*

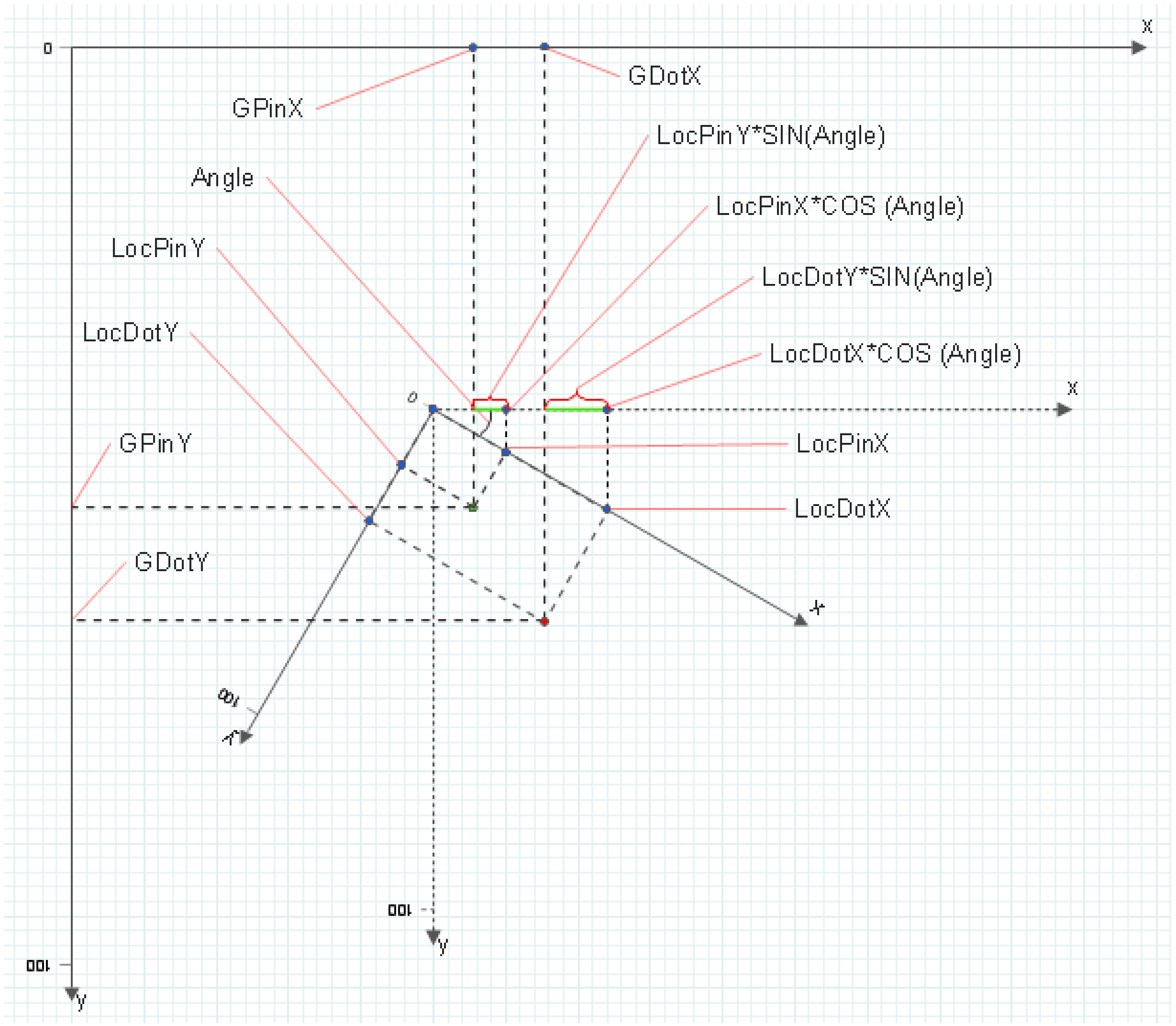
*+[(LocDotX*COS(Angle) - LocDotY*SIN(Angle)]*

Analogically GDotY:

*GDotY = GPinY –[LocPinY*COS(Angle) - LocPinX*SIN(Angle)] +*

*+[(LocDotY*COS(Angle) - LocDotX*SIN(Angle)]*

See the following figure:

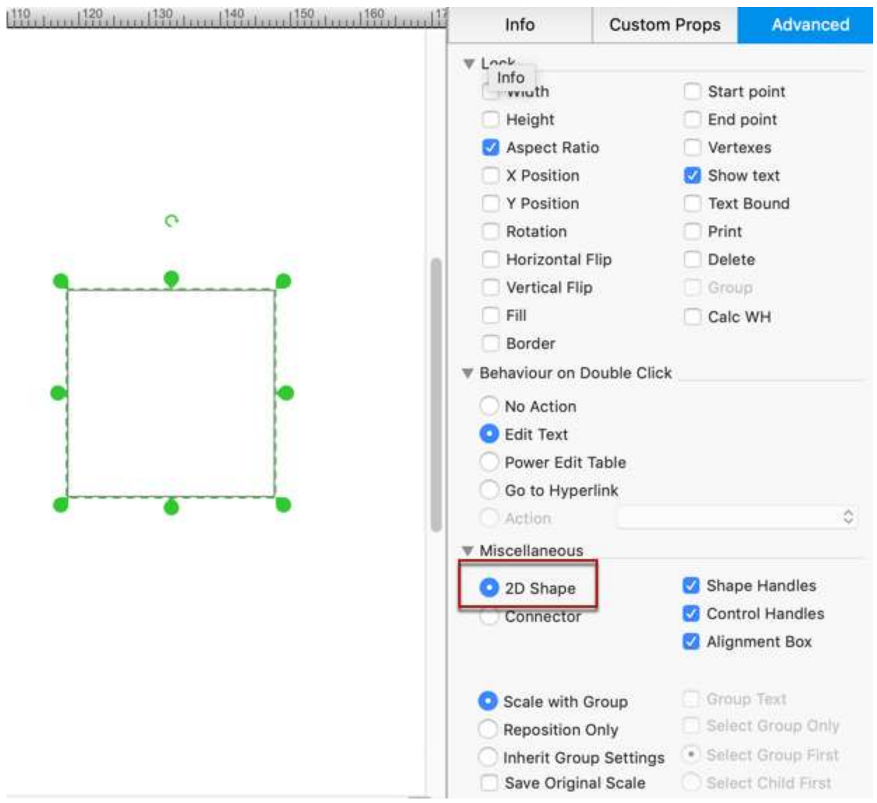


For more information on how to use the coordinates in two coordinate systems, see Lessons.

5.1.2.3. EndPoints

There are two types of objects in **ConceptDraw**: **Connectors**, and **2D Shapes**. **Connector** has a beginning and an end. **2D Shape** cannot be used as connector. It does not have start and end points. Rather, it is characterized by the width and height.

Any object can be easily converted from a **Connector** into a **2D Shape** and vice versa.



Lock

Width | Start point
Height | End point
[✓ Aspect Ratio] | Vertexes
X Position | [✓ Show text]
Y Position | Text Bound
Rotation | Print
Horizontal Flip | Delete
Vertical Flip | Group
Fill | Calc WH
Border

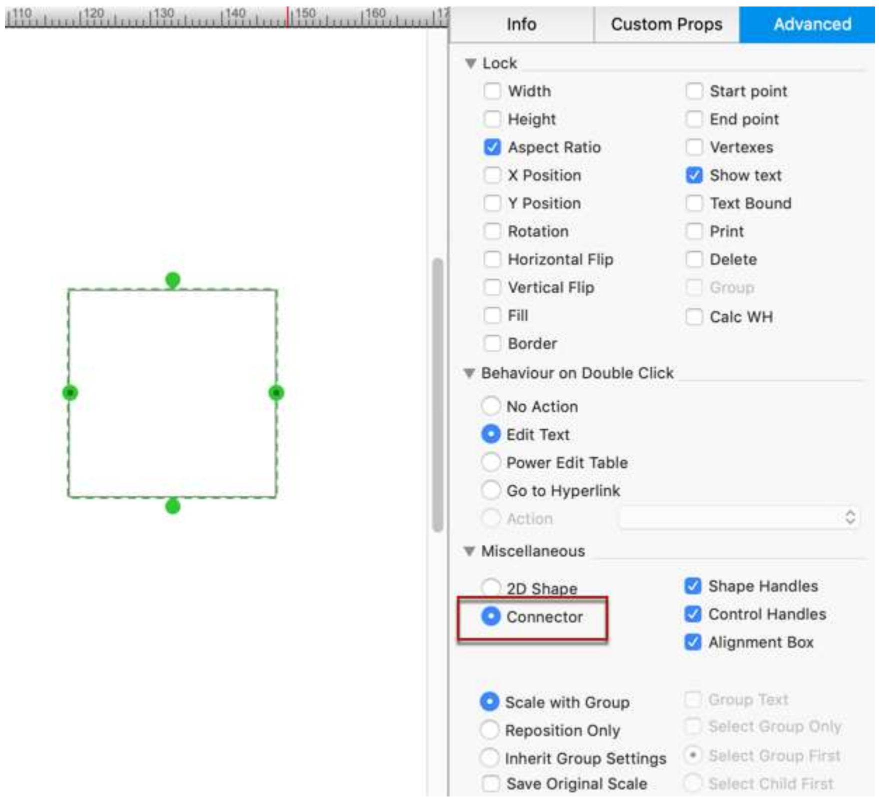
Behaviour on Double Click

No Action
[✓ Edit Text]
Power Edit Table
Go to Hyperlink
Action

Miscellaneous

[✓ 2D Shape] | [✓ Shape Handles]
Connector | [✓ Control Handles]
| [✓ Alignment Box]

[✓ Scale with Group] | Group Text
Reposition Only | Select Group Only
Inherit Group Settings | [✓ Select Group First]
Save Original Scale | Select Child First



Lock

Width | Start point
Height | End point
[✓ Aspect Ratio] | Vertexes
X Position | [✓ Show text]
Y Position | Text Bound
Rotation | Print
Horizontal Flip | Delete
Vertical Flip | Group
Fill | Calc WH
Border

Behaviour on Double Click

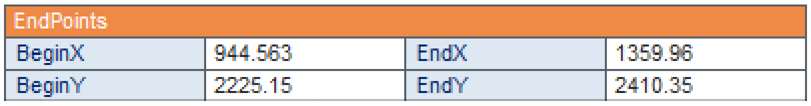
No Action
[✓ Edit Text]
Power Edit Table
Go to Hyperlink
Action

Miscellaneous

2D Shape | [✓ Shape Handles]
[✓ Connector] | [✓ Control Handles]
| [✓ Alignment Box]

[✓ Scale with Group] | Group Text
Reposition Only | Select Group Only
Inherit Group Settings | [✓ Select Group First]
Save Original Scale | Select Child First

The **EndPoints** section describes coordinates of the starting and ending points of the 1D-object, connector or smart connector relatively to the parent group. 2D object has no the **EndPoints** section.

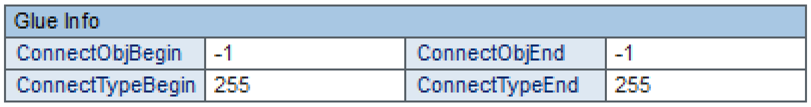


EndPoints			
BeginX	944.563	EndX	1359.96
BeginY	2225.15	EndY	2410.35

5.1.2.4. GlueInfo

This section is also inherent only for connectors (direct connectors, smart connectors, 1D-objects.). It is used to update the coordinates of the ends of the connector when the position of connected objects is changing.

The section specifies the objects by ID, which is attached to the connector **ConnectObjBegin** and **ConnectObjEnd** and the type of connection: **ConnectTypeBegin** and **ConnectTypeEnd**.



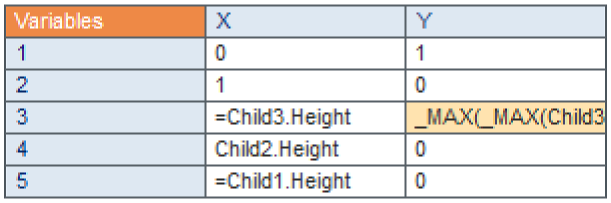
Glue Info			
ConnectObjBegin	-1	ConnectObjEnd	-1
ConnectTypeBegin	255	ConnectTypeEnd	255

5.1.2.5. Variables

You can add the internal variables to an object. They apply when the result of the same calculations need to be used in the different cells of the object's parameters table. Note that the additional variables are also convenient to identify key parameters of the object.

Variables section describes additional variables of the object. It is optional.

Each row contains two numeric variables, or the result of **X** and **Y** cells formula calculations.



Variables	X	Y
1	0	1
2	1	0
3	=Child3.Height	_MAX(_MAX(Child3
4	Child2.Height	0
5	=Child1.Height	0

5.1.2.6. Controls

The section describes the control points of the object. It requires the coordinates of the point (**X** and **Y**), and coordinates of the end of the line (**XDyn** and **YDyn**), which follow the point while it moves. Also the text for hint (**Comment**) needed. But perhaps the most interesting are the **XBehavior** and **YBehavior** properties that determine the behavior of the control point when you resize the object. The Check Point can do following:

- Move proportionally to the control frame or maintain distance relatively to any (left, right, top, bottom) edge of the frame, or to its center.
- Enable /disable the moving of the control point relative to the one of the axes.
- Switch between visible and invisible

Controls	X	Y	XDyn	YDyn	XBehaviour	YBehaviour	Comment
1	Width*0.5	_IF(_OR(Controls.Y	Width/2	0	1	2	"Reposition Upper"
2	Width*0.5	_IF(_OR(Controls.Y	Width/2	Height	1	2	"Reposition Lower"

The Controls section is optional because an object may have no control points.

5.1.2.7. Connect

In the Objects chapter there was described the ability to join objects, using connectors. Also there was mentioned that the user can create the custom connect points. Connect section describes the coordinates of the custom object's anchor points.

Connect	X	Y
1	Width*0.5	0
2	Width*0.5	Height*0.5
3	Width*0.6875	0
4	Width*0.3125	Height
5	Width	Height*0.25
6	0	Height*0.75

The **Connect** section is optional because an object may have no connect points.

5.1.2.8. Actions

The section is designed to create and describe the custom Action menu for the object. Each menu item has the following parameters:

- The action, given by formula – **Action**.
- The menu item to run the action - **Menu.**
- The hint in the status bar- **Prompt**.
- The detection of the status – **Checked**.
- The menu availability status - **Disable**.

Actions	Action	Menu	Prompt	Checked	Disabled
1	_SETF("variables.x2	"Triangle"	""	_IF(Variables.X2=3;	Variables.X1>=10
2	_SETF("variables.x2	"Square"	""	_IF(Variables.X2=4;	FALSE
3	_SETF("variables.x2	"Pentagon"	""	_IF(Variables.X2=5;	FALSE
4	_SETF("variables.x2	"Hexagon"	""	_IF(Variables.X2=6;	FALSE

The **Actions** section is optional because an object may have no actions.

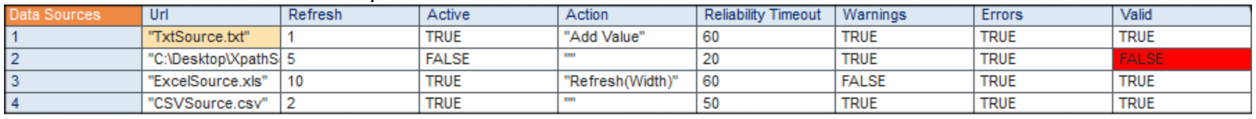
5.1.2.9. Data Sources

The section is designed to create an object data source control and work with them.

The fields of Data Sources section must contain the following data:

- The path to the source (full or relative to the document) – **URL**.
- The time lag for updating data from the source (seconds) – **Refresh**.

- The activity of the link to the source – **Active**.
- A function that is called when you update data (tabular or **CDBasic** function) - -**Action.**
- Time lag at which the connection to a data source is checked out - **Reliability Timeout**.
- Specify whether to show the object icons that indicate about the warnings and errors that arise when working with the source - **Warnings**, **Errors**.
- The correctness of the path to the data source file - **Valid**.



Data Sources	Url	Refresh	Active	Action	Reliability Timeout	Warnings	Errors	Valid
1	"TxtSource.txt"	1	TRUE	"Add Value"	60	TRUE	TRUE	TRUE
2	"C:\Desktop\XpathS	5	FALSE	""	20	TRUE	TRUE	FALSE
3	"ExcelSource.xls"	10	TRUE	"Refresh(Width)"	60	FALSE	TRUE	TRUE
4	"CSVSource.csv"	2	TRUE	""	50	TRUE	TRUE	TRUE

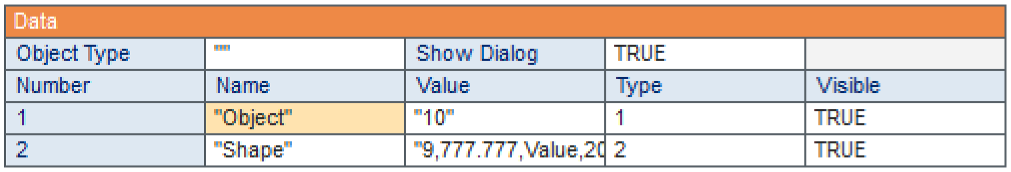
The section appears if the object has at least one data source.

5.1.2.10. Data

The section is designed to store data.

The section has as many rows as the data contained in the object.

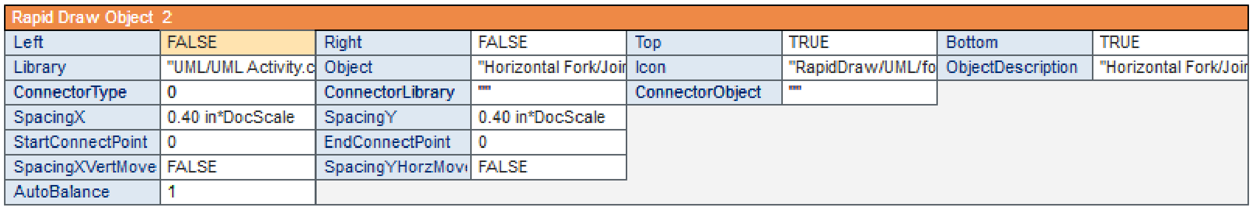
You can specify a data name and a value (or set of values, separated by commas); the data type (string, integer, float, etc.) and determine the visibility of the Value field in the "Values" dialog.



Data				
Object Type	""	Show Dialog	TRUE	
Number	Name	Value	Type	Visible
1	"Object"	"10"	1	TRUE
2	"Shape"	"9,777.777,Value,20	2	TRUE

5.1.2.11. RapidDraw Object

This section describes the settings of the **RapidDraw** object that can be constructed from the current object. An object can have the multiple RapidDraw objects, so **RapidDrawObject** sections can also be a few. All of them are numbered, for example: **RapidDrawObject 1**, **RapidDrawObject 2**, etc.



Rapid Draw Object 2							
Left	FALSE	Right	FALSE	Top	TRUE	Bottom	TRUE
Library	"UML/UML Activity.c	Object	"Horizontal Fork/Joir	Icon	"RapidDraw/UML/fo	ObjectDescription	"Horizontal Fork/Joir
ConnectorType	0	ConnectorLibrary	""	ConnectorObject	""		
SpacingX	0.40 in*DocScale	SpacingY	0.40 in*DocScale				
StartConnectPoint	0	EndConnectPoint	0				
SpacingXVertMove	FALSE	SpacingYHorzMov	FALSE				
AutoBalance	1						

The fields of **RapidDraw** section must contain the following data:

- The sides of the parent object, where the RapidDraw object can be build - **Left, Right, Top, and Bottom**.
- ConceptDraw library where the object is located (full or relative path to it) – **Library**.
- The name of the object in the library - **Object**.
- The path to the icon for the object to be displayed in the control element of the RapidDraw parent – **Icon.**
- The hint popups when you hover on the icon - **ObjectDescription**.
- The type of connector that links the parent and the current RapidDraw object (0 - smart connector, 1 – direct connector, 2 - library object) - **ConnectionType** .

- The path and the name of the library object indicated as connector – **ConnectorLibrary** and **ConnectorObject.**
- The distance between the objects horizontally and vertically (in mm or given by formula) - . **SpacingX** and **SpacingY.**
- The numbers of the anchor points for the connector of the parent object (0 - to the middle of the object, -1 ... -4 - to the sides of the object) – **StartConnectPoint.**
- The numbers of the anchor points for the connector of the current RapidDraw object (0 - to the middle of the object, -1 ... -4 - to the sides of the object) – **EndConnectPoint.**
- The displacement along the X axis and Y-axis of the object - **SpacingXVertMove** and **SpacingYHorzMove.**
- Determine the side relative to the parent object, where the RapiDraw object will be constructed (0 - left, 1 - alternately right and left) - **AutoBalance**.

5.1.2.12. Custom Properties

This section contains parameters of the object specified by the user.

The fields of Custom Properties section contain the following data:

The short name – **Label**.

The hint – **Prompt**.

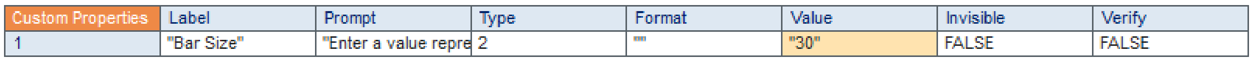
The type of the custom property **Type**.

The type of controller used in its dialog (for example: Text field, drop-down list etc) – **Format.**

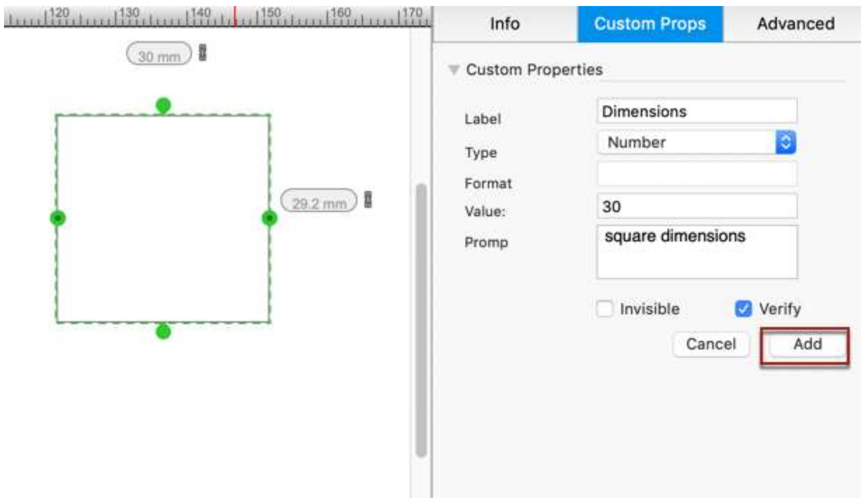
The parameter's value – **Value.**

The visibility status – **Invisible.**

The verification while object loading - **Verify**.



Custom Properties	Label	Prompt	Type	Format	Value	Invisible	Verify
1	"Bar Size"	"Enter a value repre	2	""	"30"	FALSE	FALSE



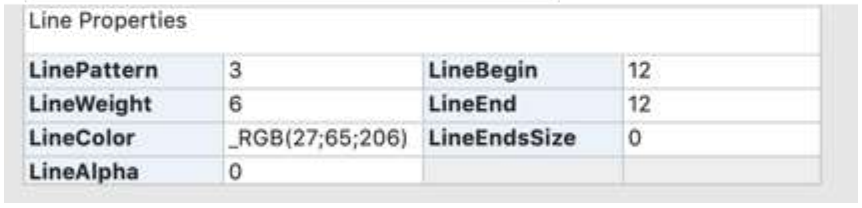
Custom Properties

Label: Dimensions
Type: Number
Format:
Value: 30
Promp: square dimensions
☐ Invisible ☑ Verify

The section is optional.

5.1.2.13. Line Properties

This section contains the variables responsible for the appearance of lines that make up the object: the color (in ConceptDraw it can be specified by index in the palette of the document in RGB or CMYK), transparency in percent (0 - completely transparent, 100 - fully transparent) thickness, type of dash line. Other variables define the appearance and size of the arrows, which can end geometries: the size of arrows and types for start and end arrows separately.



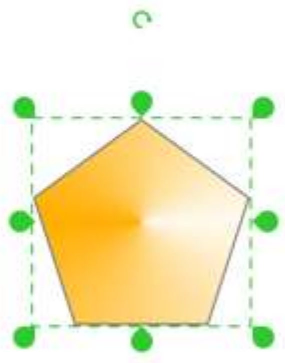
Line Properties

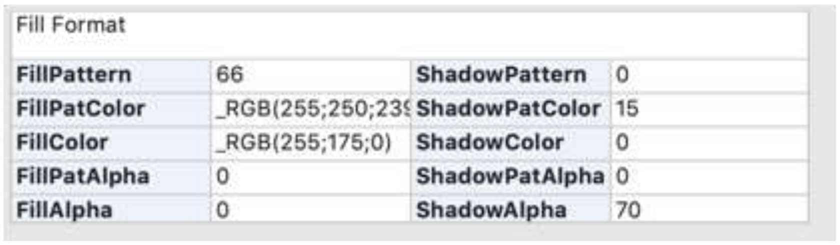
LinePattern	3	**LineBegin**	12
LineWeight	6	**LineEnd**	12
LineColor	_RGB(27;65;206)	**LineEndsSize**	0
LineAlpha	0		

You can edit these properties using the **ShapeStyle** floating dialog or the **ShapeStyle** group on the **Home** tab.

5.1.2.14. Fill Format

The section describes the properties of the object fill and shadow: their pattern and color components. Values of Pattern: 0 - no fill, 1 - solid color, from 2 to 73 - the index of the pattern. FillPatColor - the color of pattern, FillPatAlpha - its transparency. FillColor and FillAlpha - the main color of the object and its transparency. Similar properties have the shadow fill.





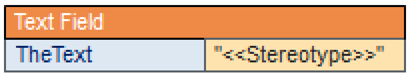
Fill Format

FillPattern	66	**ShadowPattern**	0
FillPatColor	_RGB(255;250;23	**ShadowPatColor**	15
FillColor	_RGB(255;175;0)	**ShadowColor**	0
FillPatAlpha	0	**ShadowPatAlpha**	0
FillAlpha	0	**ShadowAlpha**	70

You can edit these properties using the **ShapeStyle** floating dialog or the **ShapeStyle** group on the **Home** tab.

5.1.2.15. TextField

A separate section, consisting of only one field is used to store the text of the object. The text must be enclosed in double quotation marks.



Text Field	
TheText	"<<Stereotype>>"

There are many sections associated with different settings of the text. All of them appear only when the object is associated with the text.

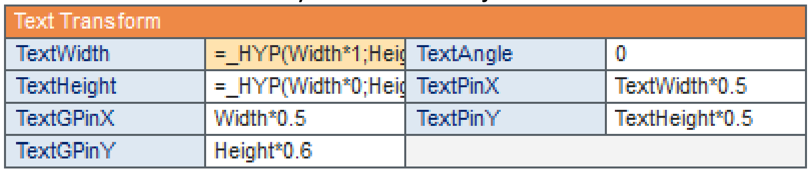
5.1.2.16. Text Transform

This section contains parameters that determine location and size of the text frame.

The section Transform describes two coordinate systems that are used in calculating the parameters of the object: the inner system of coordinates and the coordinate system of the "parent" group.

To determine the position of the text another system of coordinates, based on the text frame, is used. It is associated with the coordinate system of the object in the same manner as the latter is associated with the coordinate system of the "parent" group:

- *(TextGPinX, TextGPinY)* determine the center of rotation of the text frame in the coordinate system of the object
- *(TextPinX, TextPinY)* – relative to the text frame
- *Angle* - the angle of rotation of the coordinate system associated with a text frame, relatively to the coordinate system of the object.

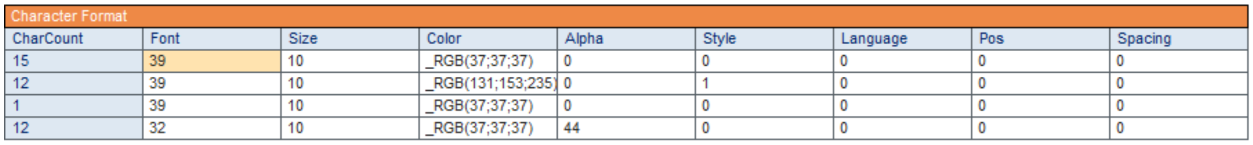


Text Transform			
TextWidth	=_HYP(Width*1;Heig	TextAngle	0
TextHeight	=_HYP(Width*0;Heig	TextPinX	TextWidth*0.5
TextGPinX	Width*0.5	TextPinY	TextHeight*0.5
TextGPinY	Height*0.6		

The section appears in the properties table of the object, only if this object is associated with the text.

5.1.2.17. Character Format

The text of the object can consist from blocks with the same set of formatting styles (these include: font (**Font**), its size (**Size**), color (**Color**), transparency (**Alpha**), font style (**Style**), a set of symbols (**Language**), position relative to the base line - the upper subscript (Pos), distance between letters (**Spacing**)). **Character Format** section describes these blocks, each on a separate line. The header line shows the number of characters in the block (**CharCount**).



Character Format								
CharCount	Font	Size	Color	Alpha	Style	Language	Pos	Spacing
15	39	10	_RGB(37;37;37)	0	0	0	0	0
12	39	10	_RGB(131;153;235)	0	1	0	0	0
1	39	10	_RGB(37;37;37)	0	0	0	0	0
12	32	10	_RGB(37;37;37)	44	0	0	0	0

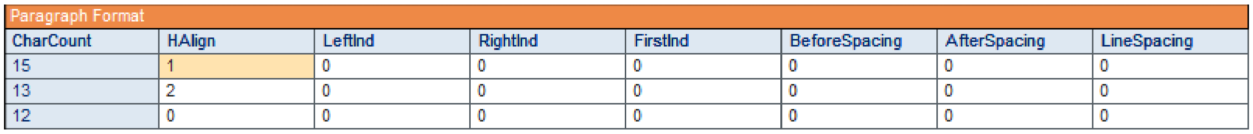
If the object has text, a table contains the **Character Format** section with at least one line.

5.1.2.18. Paragraph Format

This section determines the paragraph parameters, such as: alignment, indentation, and all kinds of line spacing. Lines in this section reflect the same sequence of formatted paragraphs. Each of these paragraphs is described by one row in the section. Follow the links below to learn the details .

AfterSpacing The distance between this paragraph and the one below. **BeforeSpacing** The distance between this paragraph and the one above. **Count** Read Only. Returns the number of characters in the paragraph. **FirstInd** The first line indent value. **HAlign** The horizontal alignment type for the paragraph.

LeftInd The distance all lines of text in a paragraph are indented from the left margin of the text block. **LineSpacing** The distance between one line of text and the next. **RightInd** The distance all lines of text in a paragraph are indented from the right margin of the text block.



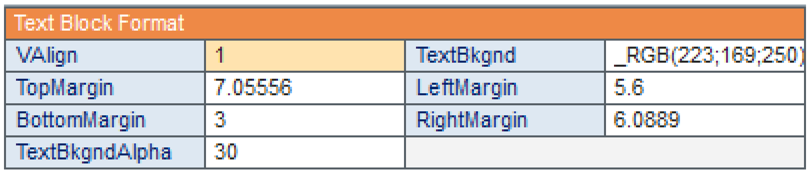
Paragraph Format							
CharCount	HAlign	LeftInd	RightInd	FirstInd	BeforeSpacing	AfterSpacing	LineSpacing
15	1	0	0	0	0	0	0
13	2	0	0	0	0	0	0
12	0	0	0	0	0	0	0

If the object has text, a table contains the **ParagraphFormat** section with at least one line.

5.1.2.19. Text Block Format

The section describes the properties of the text object as a whole. These include: vertical alignment, margins, background color, transparency and background color.

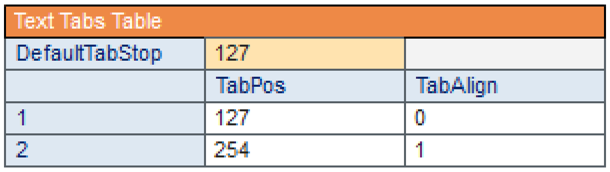
VAlign Vertical alignment type of the text within the text block. **TopMargin** The top margin of the text block. **BottomMargin** The bottom margin of the text block. **LeftMargin** The left margin of the text block. **RightMargin** The right margin of the text block. **TextBkgnd** Read-only. Text block background color. **DefTabStop** The default tab stop distance from the left edge of the text block.



Text Block Format			
VAlign	1	TextBkgnd	_RGB(223;169;250)
TopMargin	7.05556	LeftMargin	5.6
BottomMargin	3	RightMargin	6.0889
TextBkgndAlpha	30		

5.1.2.20. Tabs Table

The section describes the position and alignment of the tab stops in the object (the default value, as well as features of specific positions). **Align** Determines the alignment of the text with respect to the tab stop. **Pos** The distance between the tab stop position and the left edge of the text block, where this tab stop is located.



Text Tabs Table		
DefaultTabStop	127	
	TabPos	TabAlign
1	127	0
2	254	1

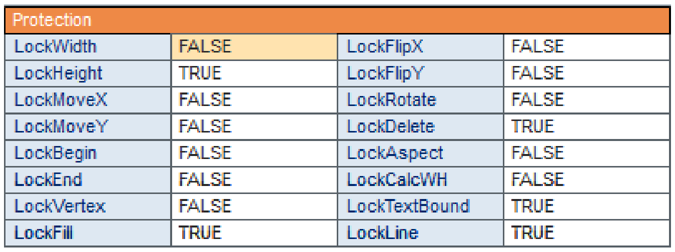
5.1.2.21. Protection

In this section you can constrain the certain user actions on the current object by using mouse. However, user can modify data through the table of parameters or through some interactions (for example, by changing the size of the group that owns this object). The Control points, which are locked for changes looks like gray locks.

You can lock the width and height of the object. You can prevent disproportionate changes of the object’s size. Also you can lock the moving on an axis, rotation, deletion, so as changes of fill and line properties.

LockAspect A flag that protects the shape from unproportional resizing. **LockBegin** A flag that protects the begin point of a 1D-shape from repositioning with the mouse. **LockCalcWH** A flag that specifies whether to update the alignment box size if the coordinates of the shape's vertices have been changed. **LockConnector** A flag that doesn't allow the smart connector to re-route automatically. **LockDelete** A flag that protects the shape from deleting. **LockEnd** A flag that protects the end point of a 1D-shape from repositioning with the mouse. **LockFlipX** A flag that protects the shape from flipping horizontally. **LockFlipY** A flag that protects the shape from flipping vertically. **LockHeight** A flag that protects the shape's height when the shape is resized. **LockMoveX** A flag that protects the shape from horizontal repositioning. **LockMoveY** A flag that protects the shape from vertical repositioning. **LockRotate** A flag that protects the shape from rotation. **LockTextBound** A flag that protects the shape fiom the cases when text does not fit within the object. **LockVertex** A flag that protects the vertices from modifying with the mouse. **LockWidth** A flag that protects the shape's width when the shape is resized.

The **LockLine** field locks the changes of the object’s line properties: color, transparency, width, arrows type. The **LockFill** field locks the changes of the object’s fill.



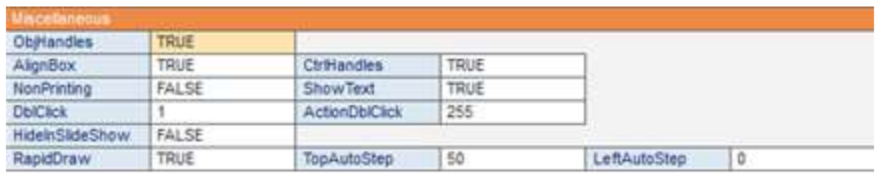
Protection			
LockWidth	FALSE	LockFlipX	FALSE
LockHeight	TRUE	LockFlipY	FALSE
LockMoveX	FALSE	LockRotate	FALSE
LockMoveY	FALSE	LockDelete	TRUE
LockBegin	FALSE	LockAspect	FALSE
LockEnd	FALSE	LockCalcWH	FALSE
LockVertex	FALSE	LockTextBound	TRUE
LockFill	TRUE	LockLine	TRUE

5.1.2.22. Miscellaneous

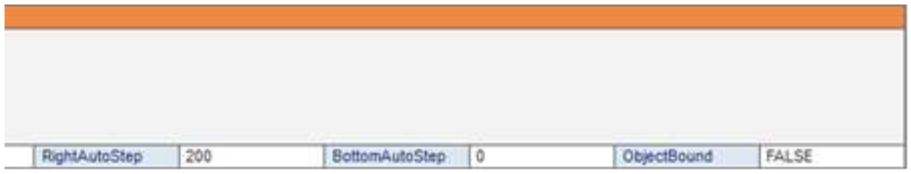
This section combines the other parameters. These parameters are responsible for the appearance and behavior of the object. Here you can determine such parameters as visibility of the control frame while selecting the object. ((**ObjHandles**), the conduct points (**AlignBox**) and the control points (**CtrHandles**).

Also you can specify parameters of the text displaying (**ShopwText**) , the object printing (**NonPrinting**) and presenting (**HideInSlideShow**).

In this section you can set the action by double-click (**DblClick** and **ActionDblClick**)



Miscellaneous					
ObjHandles	TRUE				
AlignBox	TRUE	CtrlHandles	TRUE		
NonPrinting	FALSE	ShowText	TRUE		
DblClick	1	ActionDblClick	255		
HideInSlideShow	FALSE				
RapidDraw	TRUE	TopAutoStep	50	LeftAutoStep	0



RightAutoStep	200	BottomAutoStep	0	ObjectBound	FALSE

Value in the **RapidDraw** field is set to TRUE if the object supports a quick draw. The following fields lets you customize the controls **RapidDraw**: You can specify the distance between **RapidDraw** arrows from all sides of the object (in units of the document), and the field **ObjectBound** is responsible for method determining the boundaries of the object (in his control box or on the boundary geometry of the object).

5.1.3.Table Compiler Function

In addition to drawing up a simple expression of the variables and operators you can use the built-in spreadsheet functions. Thematically, they can be divided into the following categories:

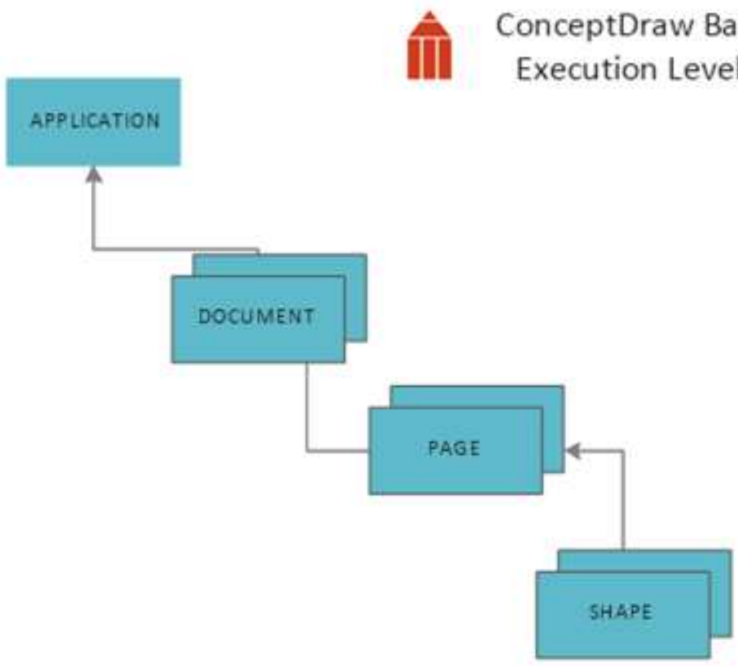
- Mathematic
- Trigonometric
- Logic
- Function of conversion and rounding
- Text
- Functions of Date and Time
- Functions of page processing
- Functions to call CD Basic subprograms
- Functions to calculate the coordinates of text of the step-type connector
- Functions to calculate the coordinates of the start and end points of connectors.
- Function for working with named styles
- Functions for working with color
- Functions for working with data sources
- Function of assigning value to a variable .

For more information on embedded table functions see Appendix 1. – "Functions and operators of the object properties table.".

5.2. Managing Object Data through CDBasic script.

The application, document, page and even object can contain a CDBasic script. This means that any document, page or ConceptDraw object, can contain some program written in ConceptDraw Basic script.

Script at any level is compiled and run at startup of the corresponding object. Initially, the application is loaded. The primary level of the script execution is the Application level. When you download a document, first run a Document script, then Page script and finally the Shape script.



The scripts of the any level consist from global execution area and a set of custom procedures that determine its performance in the local areas.

In the global area the global variables are described. Also custom and external procedures should be declared and defined there. In addition the global area contains the code to be executed immediately at startup.

Variables and named constants defined in the global area are visible in all custom procedures defined in the code, below the declaration.

Local areas are implementing the custom procedures execution. The definition of custom procedures begins with the instructions Sub or Function, and end with the instructions, EndSub or EndFunction correspondingly. The variables defined in the local area are visible only within this area. This enables using the same local variables and named constants in the different procedures. Any variable is visible down the code from its definition to the end of scope.

The ConceptDraw Basic virtual machine starts script execution from the instructions of the global area. Declaration and definition of the procedures are ignored, because they should be implemented only by call. Following the instructions of the global area or by the Stop instruction, the program goes into standby mode, remaining in residence.

ConceptDraw Basic supports the possibility to connect external modules with the code, written on ConceptDraw Basic, using the built-in directive # Include. This allows you to create different external libraries for procedures.

When you compile, the code will be integrated instead of the line with # Include directive. Thus, the structure of the script CDBasic is linear (at least within the same level of performance). In the inserted module, all the variables and procedures that are defined above the insert point are shown. Starting with the line following the connection of the module, its language constructions are visible.

ConceptDraw lets you create different versions of code for different platforms, using the mechanism of conditional compilation. To do this, there is a preprocessor directive `#If...#Else...#Endif`

```
#IfTargetBoolean

[instructions]     ' code, specific for the current operation
system

[#Else

[instructions]]     ' code, specific for the other operation
system

#EndIf
```

`TargetBoolean` can be set to one of two predefined constants: `Target_MacOS` or `Target_Win32`.

CDBasic allows you to work with variables without declaring them in advance and not caring about their type (at least for as long as we are not talking about arrays and objects). The appropriate variable is created while first assignment of values to the new identifier. These variables are of Variant type. They support the values of any type: integer and fractional numbers, dates or strings. At any time a variable of this type has a specific subtype (Integer, Boolean, String ... or a reference to any object).

When you assign a data to variable you do not need to take care of their type, the variable will go to the correct subtype by its own.

```
v = "name"     ' The variable v of the Variant type created. When
assigned got a String subtype and value "name"

 v = 2         ' v got a Long subtype and value 2
```

In addition to the variables of type Variant, you can declare variables of other types of fixed, such as:

- Boolean
- Boolean -
- integer types
- Byte, Integer and Long
- types for storage of real numbers
- Single and Double
- type for storing date and time
- Date
- string types
- String (variable-length string), String * n (FixStr, a string of fixed length)
- objects
- Application, Document, Shape, ...

Details of functions and objects ConceptDraw Basic is available in Appendix 2. , CDBasic reference

5.2.1 The table Fields changes by CDBasic script

CDBasic allows to get access to any field of the table of properties of object. The whole set of methods of object of Shape is for this purpose intended.

1. To get the value of any field of the object type use a *Get...Property()* methods. Indicate the type of property Instead of dots (*GetBooleanProperty(), GetByteProperty(), GetStringProperty()*). To select the desired field use tease three arguments: *propTag, num, geom*. Here *propTag is*- a tag according to the property name (e.g. *CDPT_GEOMETRY_X* - X corresponds to a column section Geometry), and *num* and *geom* determine the number of the current property among the same names in collections.

2. Set the field value with a help of the set of similar methods *Set...Property()*.

3. **CDBasic** provides an opportunity to work not only with the values of table fields, but with the tabulated formulas. Methods of object Shape allows to obtain information about the formulas:

IsDefaultFormula()	Is there a default formula in the fieldc
IsNullFormula()	Is there a formula or constant
GetPropertyFormula()	View of the certain formula (e.g., "=Width*0.4")

Or change it:

SetPropertyFormula()	Set the custom formula
SetDefaultFormula()	Set the default formula
SetNullFormula()	Delete formula (the value remain)

4. Field in a table object can be divided into two big categories: those correspond to the properties of the Shape, others are available through the Shape object collections of other objects. The latter include optional sections of the table fields (except the Text Field and Text Transform), which the user can add by his own and in any quantity. Most of the mandatory fields in the table is reflected in the object properties (*Width, GPin, FillColor*...). To manage their values you can use such methods as *Get...Property(), Set...Property()*. Also they can be accessed directly through the properties of an object *Shape*. But this applies only to the values. The approach for work with tabular formulas is common for both types of fields.

5. There is one important difference when working with these two types of fields from **CDBasic**. If you make changes through the object properties *Shape* , the corresponding fields are recalculated automatically. But the changes through such methods as *Set...Property(), Set...Formula()* do not do this . They do not fall in line for recalculation (if there is a formula), and the resulting object does not change and is not redrawn. To "infrom" the Object that such a field and associated fields should be recalculated, you need to report this manually. There two object *Shape* methods for this purpose: *PropertyChanged() and RecalcProperty().*. The first one simply tells the application that the field is changed, and therefore the associated fields should be recalculated. It is used when the field does not contain formulas and there is nothing to recalculate in the field. Just the associated ones. The second requires the field recalculating according formula. If this field will fall into line for recalculation, the associated fields will be processed automatically. There is no needs to call *RecalcProperty()*

If the method *PropertyChanged()* or *RecalcProperty()* was called somewhere within a couple *StartRebuild() - EndRebuild()*, the actual recalculation will be done by calling the document EndRebuild (), otherwise the recalculation will be executed immediately.
If one of these methods has been called from a custom procedure, which in turn was called from the table, then the conversion caused by *PropertyChanged()* or *RecalcProperty()* occurs immediately after the recalculation that caused custom procedure *Property*.

Examples (for the *Shape* level script)
Let's reflect the object horizontally

thisShape.SetBooleanProperty(True,CDPT_FLIPX)

First set the value and then the field *CDPT_FLIPX* – tag, corresponding to the field *FlipX* from section Transform.
Let's set a new formula to the field *LocPinX*
*thisShape.SetPropertyFormula("_MIN(Width,Height)*0.5",CDPT_LPINX)*
thisShape.RecalcProperty(CDPT_LPINX)
The first line sets a new formula, and the second line declares that the value of the field should be recalculated according to the new formula (as well as all associated fields).
In the example above the field LocPinX assigned a custom formula (not default). Let's show how to change the field value without default formula editing. The following rows set the value 200 into the field *Geometry.X1.*

thisShape.SetDoubleProperty(200,CDPT_GEOMETRY_X,1,1)
thisShape.SetDefaultFormula(CDPT_GEOMETRY_X,1,1)
thisShape.PropertyChanged(CDPT_GEOMETRY_X,1,1)
For fields of the **Geometry** section, except field type, the rows and geometries numbers must be specified. The first row changes the value. The second - indicates that even for this value, the formula must be default. This means that coefficient for the current view and value of the formula is needed. The third row is reported that the field value has changed.

Appendix 1. Functions and Operators Table object properties

A. 1.1. Operators of the table object's properties

In the formulas, tables can use arithmetic operators, comparison operators and logical operators.

A.1.1.1. Arithmetic Operators

Operator	The action of

^ Or **	Raise a number to a power.
	`Example:`
	`2 ** 10 bring back 1024.`

-	The change of the sign.
	`Example:`
	`-5.`
	`- (3) will give us 3.`

+	Addition of two expressions. If the logical expression returns a Boolean expression, if numeric - the result of the sum, if the string type, then merge the two lines. If one of the expressions (any) - string, the second - of any

	type, then merge the rows with non-string type conversion of expression into a string.
	`An example.`
	`Returns a 1 2`
	`2 + "some string" dastnam "2some string",`
	`"Some string" +234 return "some string234".`

-	Subtract one number from another.
	`An example.`
	`3.1 will give 2,`
	`4.7 budetravno 3.`

*****	The product of two expressions.
/	Dividing the first expression in the second. When you divide by zero error does not perform, and returns the first expression.

\	Integer division of one number by another.
	`Example:`
	`11 \ 4 2 budetravno`
	`9 \ 2 yields 4`

MOD	Divides one number by another and returns the remainder of the division.
	`Example:`
	`10 MOD 3 returns 1`
	`8 MOD 5 returns 3`
	`8 MOD 3 equals 2`

&	The operator that returns only the string type - the ***&*** operator. This operator merges two strings. If both expressions are not strings, or at least one of them, the operator converts them to string and then perform the merge.
	`Example:`
	`"Some" & "string" return "somestring",`
	`34 & "string" return "34string",`
	`45 & 56 will return the string "4556."`
	Notice. Generally speaking, this operator is not arithmetic, but referred to this group as long as it has the same priority as the operators of this group over the other operators.

A.1.1.2. Comparison operators:

Operators less than **"<",** greater than **">"** is less than or equal to **"<="** greater than or equal to **">="** equal **"="** not equal **"<>"** are used to compare two expressions.

Syntax:

<Result> = <Var1> <comparison operator> <Var2>.

A.1.1.3. Logical Operators

They can be used in formulas except the logic functions *_AND (), _IF (), _NOT (), _OR ()* and *_XOR ().*

Operator	Action
AND	A Boolean "and" the conjunction is true if both arguments are true and false in all other cases. If the parameters that are used - the number, then the bitwise conjunction. *Example:* *2> 3 AND 2 <5 returns TRUE* *2 AND 3 returns 2 (bitwise "and")*
EQV	The operator logicheskoyekvivalentsii two expressions. If it works with logical expressions, returns TRUE, only if both expressions are true (TRUE), or both expressions are false (FALSE). With the numbers of the operator operates the same way, only bit. *Truth table:* *For logical expressions.*

A	*B*	*A EQV B*
True	*True*	*True*
True	*False*	*False*
False	*True*	*False*
False	*False*	*True*

For arithmetic expressions

a	*b*	*a EQV b*
A	*A*	*A*
A	*0*	*0*
0	*A*	*0*
0	*0*	*A*

Operator	Action
IMP	Operator logical implication on two expressions (the investigation). For bitwise arithmetic works. *Truth table:* *For logical expressions.*

A	*B*	*A IMP B*
True	*True*	*True*
True	*False*	*False*
False	*True*	*True*
False	*False*	*True*

For arithmetic expressions.

a	*b*	*a IMP b*
A	*A*	*A*
A	*0*	*0*
0	*A*	*A*
0	*0*	*A*

NOT or!

The operator of logical negation. When working with numbers deystvuetpobitovo.
Entries and ***NOTA! A*** - mean the same thing.
Truth table:
For logical expressions

A	*NOT A*
True	*Fa se*
False	*True*

For arithmetic expressions.

a	*NOT a*
A	*0*
0	*A*

OR

Logical operator "or" disjunction. Returns true (TRUE), when at least one expression is true, otherwise returns false (FALSE). For bitwise arithmetic works.
Truth table:
For logical expressions.

A	*B*	*A OR B*
rue	*True*	*True*
True	*False*	*True*
False	*True*	*True*
False	*False*	*False*

For arithmetic expressions.

a	*b*	*a OR b*
A	*A*	*A*
A	*0*	*A*
0	*A*	*A*
0	*0*	*0*

XOR

The operator exclusive "or". Returns true (TRUE), only when one of two expressions is true, and the other is false.
Truth table:
For logical expressions.

A	*B*	*AXORB*
True	*True*	*False*
True	*False*	*True*
False	*True*	*True*
False	*False*	*False*

For arithmetic expressions.

a	b	a XOR b
A	A	0
A	0	
0	A	A
0	0	0

A.1.1.4. Precedence of operators.

The expressions in the formulas can be separated by brackets to make it easier to understand on what Element of the expression is valid or that the operator. If you know the priority of the operators, we can simplify the job without placing a large number of unnecessary parentheses in an expression.

There are three groups of operators: arithmetic operators, comparison operators, logical operators. They are divided into groups so not only because of their destination, but also in terms of priority. The highest priority is given to the arithmetic operators, followed by the comparison operators and logical operators have the lowest priority and are executed after all the operators of the first two groups (if not placed brackets). Within these groups of operators are also distributed by priority.

Arrange operators in the groups in descending order of their priority from top to bottom, and in descending order of priority groups from left to right:

Arithmetic Operators	Comparison Operators	Boolean Operators
^ ** And a	=	NOT
enaznaka cm "-"	<>	AND
* /	<	OR
\	>	XOR
MOD	<=	EQV
+ -	> =	IMP
&		

A.1.2. Table Compiler Functions

In the properties of the formulas you can use the built-in spreadsheet functions. Thematically, they can be divided into the following categories:

All table functions begin with an underscore, and are written in big letters. For example: *_CALLTHIS (), _MIN (), _LN ().* But if you can set down the underscore character. Register is also not important. The editor automatically converts your input to meet these requirements.

By category:

Math:

- _ABS
- _CENTERX
- _CENTERY

- _CIRCLE_CENTERX
- _CIRCLE_CENTERY
- _CIRCLES3RD_X
- _CIRCLES3RD_Y
- _CUT
- _ELLIPSE_ANGLE
- _ELLIPSE_ASPECT
- _FABS
- _GRAVITY
- _HYP
- _LG10
- _LN
- _LOCALX
- _LOCALY
- _MAX
- _MIN
- _MOD
- _POW
- _RAND
- _SIGN
- _SQRT
- _WORLDX
- _WORLDY

Trigonometric:

- _ACOS
- _ASIN
- _ATAN
- _ATAN2
- _COS
- _COSH
- _PI
- _SIN
- _SINH
- _TAN
- _TANH

Logic:

- _AND
- _IF

- _NOT
- _OR
- _XOR

Function of conversion and rounding:

- _ANG360
- _DEG
- _RAD
- _ROUND
- _FLOOR

Text:

- _CHR
- _EVALTEXT
- _FILENAME
- _FULLFILENAME
- _MEASURE
- _SCALE
- _TEXTHEIGHT
- _TEXTLEFT
- _TEXTLENGTH
- _TEXTRIGHT
- _TEXTWIDTH
- _VALTOTEXT
- _VALTOTEXTMES

Date and time:

- _DATE
- _TIME

Functions of page processing :

- _PAGENUMBER
- _PAGESCOUNT
- _PAGEWIDTH

Functions for a call of subprogrammes on ConceptDrawBasic.

- _CALLTHIS
- _CALLTHIS_1ARG

- _CALLTHIS_2ARGS

Functions to calculate the coordinates of the text step connector.

- _SMARTCONNECTORTEXTX
- _SMARTCONNECTORTEXTY

Functions to calculate the coordinates of the starting and ending points of the connector.

- _CONNECTBEGINX
- _CONNECTBEGINY
- _CONNECTENDX
- _CONNECTENDY

Functions for working with named styles

- *Functions for work with style of lines*
    - _STYLED_ENDSSIZE
    - _STYLED_LINEBEGIN
    - _STYLED_LINECOLOR
    - _STYLED_LINEEND
    - _STYLED_LINEPATTERN
    - _STYLED_LINEWEIGHT
- *Functions for work with filling and a shadow*
    - _STYLED_FILLBGNDALPHA
    - _STYLED_FILLCOLOR
    - _STYLED_FILLCOLORBGND
    - _STYLED_FILLCOLORFGND
    - _STYLED_FILLFGNDALPHA
    - _STYLED_FILLPATCOLOR
    - _STYLED_FILLPATTERN
    - _STYLED_PENALPHA
    - _STYLED_PENCOLOR
    - _STYLED_PENPATTERN
    - _STYLED_PENWEIGHT
    - _STYLED_SHADOWBGNDALPHA
    - _STYLED_SHADOWCOLOR
    - _STYLED_SHADOWCOLORBGND
    - _STYLED_SHADOWCOLORFGND
    - _STYLED_SHADOWFGNDALPHA
    - _STYLED_SHADOWPATCOLOR
    - _STYLED_SHADOWPATTERN

- *Functions for work with fonts*
  - _STYLED_FONTALPHA
  - _STYLED_FONTCHARLANG
  - _STYLED_FONTCHARSET
  - _STYLED_FONTCOLOR
  - _STYLED_FONTNUM
  - _STYLED_FONTPOS
  - _STYLED_FONTSIZE
  - _STYLED_FONTSPACING
  - _STYLED_FONTSTYLE
- *Functions for work with paragraphs*
  - _STYLED_PARAAFTERINDENT
  - _STYLED_PARAAFTERSPACING
  - _STYLED_PARABEFOREINDENT
  - _STYLED_PARABEFORESPACING
  - _STYLED_PARABETWEENLINE
  - _STYLED_PARAFIRSTLINE
  - _STYLED_PARAHALIGNMENT
  - _STYLED_PARALEFTINDENT
  - _STYLED_PARALINESPACING
  - _STYLED_PARARIGHTINDENT
- *Functions for work with the text block*
  - _STYLED_TXTBKGNDCOLOR
  - _STYLED_TXTBOTTOMMARGIN
  - _STYLED_TXTDEFTABSTOP
  - _STYLED_TXTLEFTMARGIN
  - _STYLED_TXTRIGHTMARGIN
  - _STYLED_TXTTOPMARGIN
  - _STYLED_TXTVALIGN

Working with color:

- _CMYK
- _GRADCOLOR
- _HTML2RGB
- _RGB

Functions for working with data sources

- *Functions for work with CSV*
  - _CSVCOLORVALUE
  - _CSVGETCOLUMNFORKEY

- _FILETEXT
- _GETVALUE
- _GETVALUEEL

Miscellaneous:

- _DOFORCONNECTED
- _GLUETOSERVICE
- _SETF

A.1.3. Compiler Options table alphabetically with a description of

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

_ABS

_ABS (Arg)

Returns the absolute value of arg.

_ABS (Str)

Returns the string str unchanged.

Examples:
_ABS (-3) = 3
_ABS (0) = 0
_ABS (4) = 4
_ABS ("Text") = "Text"

_ ACOS

_ACOS (Arg)

Returns the arc cosine of arg (it is in the range of-pi / 2 to pi / 2).
The argument must be in the range from -1 to 1. Otherwise, the error code generated.

_ AND

_AND (Arg1; arg2)

Returns the bitwise "and";

_AND (Str1; str2)

Returns 1 - if non-empty string, 0 - if even one of them - empty.

_AND (Str; arg)
_AND (Arg; str)

Returns arg number.

Examples:
_AND (1, 0) = 0
_AND (3, 2) = 2
_AND ("Hello!"; "") = 0
_AND ("Text1"; "Text2") = 1
_AND ("Text"; 2) = 2

_ ANG360

_ANG360 (Arg)

Returns the value of the angle arg, reduced to the interval from 0 to 2 * pi.

Examples:
_ANG360 (481 deg) = 121 deg
_ANG360 (-4.5 Rad) = 1.7832 rad

_ASIN

_ASIN (Arg)

Returns the arc sine of arg (it is in the range of-pi / 2 to pi / 2).
The argument must be in the range from -1 to 1. Otherwise, the error code generated.

_ATAN

_ATAN (Arg)

Returns the arc tangent of arg (it is in the range of-pi / 2 to pi / 2).

_ATAN2

_ATAN2 (Arg1; arg2)

Returns the arctangent of a number (arg1/arg2). Unlike the _ATAN, _ATAN2 properly handle expressions with a zero value of arg2. In any case, the value is between-pi / 2 to pi / 2.

Examples:
_ATAN (1, 0) = 90 deg
_ATAN (2, 2) = 45 deg

_CALLTHIS

Function calls written in the embedded language CDBasic.

_CALLTHIS ("Proc_name")

The name of the function being called must be in quotes.

The function is invoked must have the following form in the editor CDBasic:

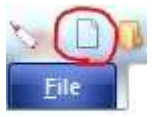
```
Function proc_name (shp As Shape) [As <Type>]
```

This variable represents the **shp** object from which the function is called.

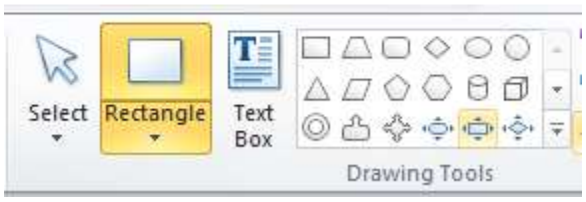
_CALLTHIS Returns the result, which returns the specified function.
The result type ***_CALLTHIS*** the same as that of the function is called.

Example of use.

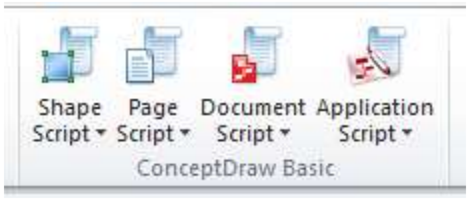
Create a new file (menu **File / NewDocument,** either by pressing the toolbar **Main).**



Select the toolbar **DrawingTools** tool and draw a rectangle.



Start the editor via the menu CDBasic **Document button in the group DocumentScript Conceptdraw Basic.**



In the editor, type the following lines:

```
Function MyProc (shp As Shape) As Byte
   Dim ss As String
   ss = InputBox $ ("Enter text for shape")
   shp.Text = ss
End Function
```

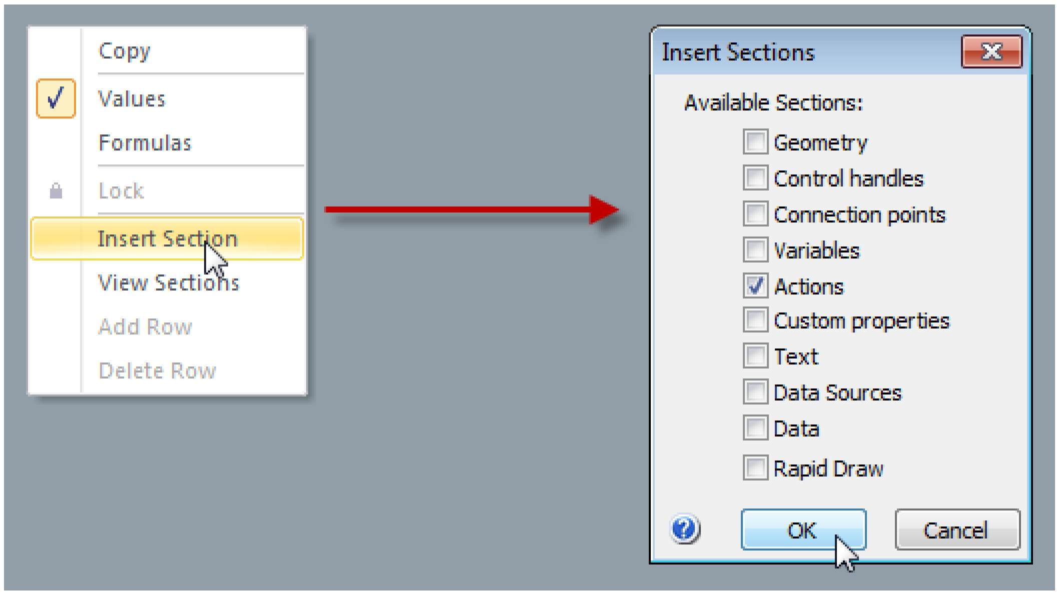
This will be the description of the function *MyProc,* which will be called from the context menu of an object by _***CALLTHIS.*** Close the editor window CDBasic.

Now add a rectangle custom context menu.

Select the rectangle you created, using the tool

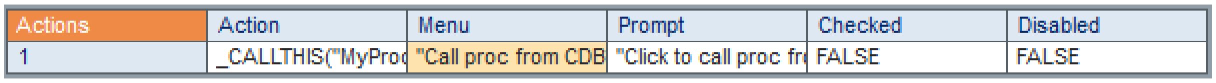
from the toolbar **DrawingTools.** Call a configuration of the object by pressing **F3** or by pressing the menu **button, Shape Power Edit.**

Add the **Actions** section of the table object parameters by selecting the context menu of any table **Insert Section,** and then in the dialog box, select the **Insert Sections Actions** and then click **OK.**



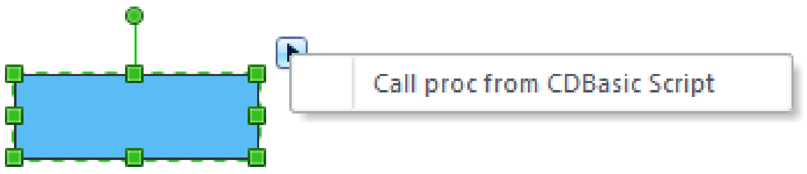
In the **Actions** section that appears in the **Action** field, enter the call of our function *_CALLTHIS ("MyProc")*

In the **Menu,** type the name of the menu item **"Call proc from CDBasic script",** in the **Prompt,** enter **"Click to call proc from CDBasic Script",** in the fields of **Checked** and **Disabled,** leave untouched to **FALSE.**



Actions	Action	Menu	Prompt	Checked	Disabled
1	_CALLTHIS("MyProc	"Call proc from CDB	"Click to call proc fr	FALSE	FALSE

Now close the property sheet object. The object appeared in selecting a button, when pressed on the buyout menu appears Action- menu.



Paragraph calling user-defined function writing on CDBasic.

_ CALLTHIS _ 1ARG

The function works similarly _CALLTHIS, except that the called function is passed a parameter.

Syntax:

_CALLTHIS_1ARG ("Proc_name"; arg1)

Here **proc_name** - the name of the called function, and arg1 - parameter passed to it.

The callee CDBasic should have the following form:

```
Function proc_name (shp As Shape, arg1 As <Type1>) [As <Type>],
```

where **shp** - the caller is a function of the object, **arg1** - an argument that should be transferred to the function.

`Notice.` It is important to match the transmitted parameter types specified in the description and function, and it is also important to specify match the return type of function caused by **_CALLTHIS_1ARG** value of the type that should bytpoluchen calling **_CALLTHIS_1ARG.**

_ CALLTHIS _ 2ARGS

The function operates similarly to **_CALLTHIS** and **_CALLTHIS_1ARG** , except that the called function language CDBasic passed two parameters.

Syntax:

CALLTHIS_1ARG ("proc_name"; arg1; arg2)

Here **proc_name** - the name of the called function, **arg1, arg2** - the parameters passed to it. The called function must be described as follows:

```
Function proc_name (shpAsShape, arg1 As <Type1>, arg2 As <Type2>) [As
<Type>]
```

Here shp - is the object that caused the function, arg1 and arg2 - the parameters passed.

Notice. As for **_CALLTHIS** and **_CALLTHIS_1ARG** important type matching the passed parameters that are used in the description of the function, and the correct type of function return values.

_CENTERX

_CENTERX ()
Returns the X coordinate of the center of the object. Under the center is meant for: - Smart Connector - the middle of the central segment, if an odd number of segments, the intersection of the two middle segments, if an even number. - For other objects - center management framework.

This function is used, for example, to position the text with intelligent connector.

_CENTERY

_CENTERY ()
Returns the Y coordinate of the center of the object. Under the center is meant for: - Smart Connector - the middle of the central segment, if an odd number of segments, the intersection of the two middle segments, if an even number. - For other objects - center management framework.

This function is used, for example, to position the text with intelligent connector.

_ CHR

_CHR (Arg)

Returns a numeric value corresponding to this value of the symbol.

Example:

_CHR (32) returns the character number 32 ("gap").

_ CIRCLE _ CENTERX

_CIRCLE_CENTERX (X1; Y1; X2; Y2; X3; Y3)
Returns the X coordinate of the center of the circle, built on three points: (X1; Y1), (X2; Y2) and (X3; Y3).

_CIRCLE_CENTERY

_CIRCLE_CENTERY (X1; Y1; X2; Y2; X3; Y3)
Returns the Y coordinate of the center of the circle, built on three points: (X1; Y1), (X2; Y2) and (X3; Y3).

_CIRCLES3RD_X

_CIRCLES3RD_X (X1; Y1; X2; Y2; H)
Returns the X coordinate of a point located at a distance H from the middle of the vector (X1; Y1) - (X2; Y2). If H - a positive number, the point is plotted on the left side of the vector, if the H - negative number - on the right side.Used to set the arc of a circle with two points and the height of the arc.

_CIRCLES3RD_Y

_CIRCLES3RD_Y (X1; Y1; X2; Y2; H)
Returns the Y coordinate of a point located at a distance H from the middle of the vector (X1; Y1) - (X2; Y2). If H - a positive number, the point is plotted on the left side of the vector, if the H - negative number - on the right side.Used to set the arc of a circle with two points and the height of the arc.

_ CMYK

Sets the color of the standard CMYK.

Syntax:

_CMYK (C; M; Y; K)

where C, M, Y, K - component in the standard CMYK.

An example.

_CMYK (100, 0, 100, 0)

give a yellow-green color.

Used for setting the colors in the parameter table (for parameters such as the section of the **FillColor Fill, LineColor LineProperties** section of the table object parameters.

_ CONNECTBEGINX

It is used in the program to calculate the coordinates of the starting point of the connector on the axis X.

Syntax:

_CONNECTBEGINX (ObjBegin; ObjEnd; TypeBegin)

Here **ObjBegin** - ID of the object, joined by top connector, **ObjEnd** - ID of the object, which joins the end of the connector. **TypeBegin** - fitting, from 1 to 4 - adherence to any of the Elementies, -1 - no connection, 5 compound with the middle any side, so that the distance was minimal.

Typically used for the needs of the translation section of ConceptDraw **EndPoints** parameters for connectors. It uses the parameters of section **GlueInfo** table settings.

Begin X is calculated as follows: **_CONNECTBEGINX (ConnectObjBegin; ConnectObjEnd; ConnectTypeBegin),** where **ConnectObjBegin, ConnectObjEnd, ConnectTypeBegin** - the parameters of section **GlueInfo.**

_ CONNECTBEGINY

It is used in the program to calculate the coordinates of the starting point of the connector on the axis Y.

Syntax:

_CONNECTBEGINY (ObjBegin; ObjEnd; TypeBegin)

Here **ObjBegin** - ID of the object, joined by top connector, **ObjEnd** - ID of the object, which joins the end of the connector. **TypeBegin** - fitting, from 1 to 4 - adherence to any of the Elementies, -1 - no connection, 5 compound with the middle any side, so that the distance was minimal.

Typically used for the needs of the translation section of ConceptDraw **EndPoints** parameters for connectors. It uses the parameters of section **GlueInfo** table settings.

BeginY calculated as follows: **_CONNECTBEGINY (ConnectObjBegin; ConnectObjEnd; ConnectTypeBegin),** where **ConnectObjBegin, ConnectObjEnd, ConnectTypeBegin-parameters** section of the table **GlueInfo.**

_ CONNECTENDX

It is used in the program to calculate the coordinates of the end point of the connector along the axis X.

Syntax:

_CONNECTENDX (ObjBegin; ObjEnd; ConnectTypeEnd)

ObjBegin - ID of the object, which joins the beginning of the connector, **ObjEnd** - ID of the object, which joins the end of the connector. **ConnectTypeEnd** - fitting, from 1 to 4 - adherence to any of the Elementies, -1 - no connection, 5 compound with the middle of what either side, so that the distance was minimal.

Typically used for the needs of the translation section of ConceptDraw **EndPoints** parameters for connectors. It uses the parameters of section **GlueInfo** table settings.

_ CONNECTENDY

It is used in the program to calculate the coordinates of the end point of the connector along the axis Y.

Syntax:

_CONNECTENDY (ObjBegin; ObjEnd; ConnectTypeEnd)

ObjBegin - ID of the object, which joins the beginning of the connector, **ObjEnd** - ID of the object, which joins the end of the connector. **ConnectTypeEnd** - fitting, from 1 to 4 - adherence to any of the Elementies, -1 - no connection, 5 compound with the middle of what either side, so that the distance was minimal.

_COS

_COS (Arg)
Returns the cosine of arg (it is in the range from -1 to 1).

_COSH

_COSH (Arg)
Returns the hyperbolic cosine of arg.

_CSVCOLORVALUE

NEW

Returns the color value written to the specified data source.

The color in the source data must be written in standard Web (eg # A180FF).

_CSVCOLORVALUE (DSNUM; NROW; NCOL; DEFCOLOR)

The arguments are:

DSNUM - number of the source data in the source list.

NROW - nomerstroki to address in a table view CSV file (numbered from 1)

NCOL - nomerstolbtsa to address in a table view CSV file (numbered from 1)

DEFCOLOR - the color value of the default output for the case of addressing the range of the table or if the data are not available.

The value is vnuzhnyh sections of one or neskolkihtablits object parameters (such as a table for the field **FillFormat FillColor** or **CustomProperties** table for the field of **Value).**

Example:

_CSVCOLORVALUE (1, 4, 5; _RGB (255, 0, 0))

_CSVGETCOLUMNFORKEY

NEW

Returns the number of columns in the specified source dannyhpri by searching by key.

_CSVGETCOLUMNFORKEY (DSNUM; KEYROW; KEYSTR)

The arguments are:

DSNUM - number of the source data in the source list.

KEYROW - nomerstroki to address in a table view CSV file (numbered from 1)

KEYSTR - a key string to search.

If the transmitted key string passed to the key column is not found, the function returns 0.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVGETCOLUMNFORKEY (1, 4; "green")

_CSVMAXELEMENT

NEW

Returns the maximum element of the string from the specified data source.

The return value is rounded to an integer.

Argumentamiyavlyayutsya:

_CSVMAXELEMENT (DSNUM; NROW; DEFVAL)

DSNUM - number of the source data in the source list.

NROW - nomerstroki to address in a table view CSV file (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value is an integer.

The value is in the right sections of one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVMAXELEMENT (1, 4, -1)

_CSVMAXELEMENTD

NEW

Returns the maximum element of the string from the specified data source.

_CSVMAXELEMENTD (DSNUM; NROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NROW - nomerstroki to address in a table view CSV file (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value can not be an integer.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVMAXELEMENTD (1, 4, -1.5)

_CSVMINELEMENT

NEW

Returns the minimum element of the string from the specified data source.

The return value is rounded to an integer.

_CSVMINELEMENT (DSNUM; NROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NROW - nomerstroki to address in a table view CSV file (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value is an integer.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVMINELEMENT (1, 1, -1)

_CSVMINELEMENTD

NEW

Returns the minimum element of the string from the specified data source.

_CSVMINELEMENTD (DSNUM; NROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NROW - nomerstroki to address in a table view CSV file (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value can not be an integer.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVMINELEMENTD (1, 2, -1)

_CSVMINROWLENGTH

NEW

Returns the minimum length of a string (of all lines) for the specified data source.

_CSVMINROWLENGTH (DSNUM)

The argument is:

DSNUM - number of the source data in the source list.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVMINROWLENGTH (2)

_CSVROWLENGTH

NEW

Returns the number of line items in the specified data source.

_CSVROWLENGTH (DSNUM; NROW)

The arguments are:

DSNUM - number of the source data in the source list.

NROW - nomerstroki to address in a table view CSV file (numbered from 1)

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVROWLENGTH (1 2)

_CSVROWMAXELEMENT

NEW

Returns the maximum element of the string from the specified data source.

_CSVROWMAXELEMENT (DSNUM; NROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NROW - nomerstroki to address in a table view CSV file (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value can not be an integer.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVROWMAXELEMENT (1, 4, -1.5)

_CSVROWMINELEMENT

NEW

Returns the minimum element of the string from the specified data source.

_CSVROWMINELEMENT (DSNUM; NROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NROW - nomerstroki to address in a table view CSV file (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value can not be an integer.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVROWMINELEMENT (1, 2, -1)

_CSVROWNUM

NEW

Returns the number of lines in the specified data source.

_CSVROWNUM (DSNUM)

The argument is:

DSNUM - number of the source data in the source list.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVROWNUM (3)

_CSVTEXT

NEW

Returns the text written in the specified data source.

_CSVTEXT (DSNUM; NROW; NCOL; DEFSTR)

The arguments are:

DSNUM - number of the source data in the source list.

NROW - nomerstroki to address in a table view CSV file (numbered from 1)

NCOL - nomerstolbtsa to address in a table view CSV file (numbered from 1)

DEFSTR - the term for the case of a default address for the output range of the table or if the data are not available.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value** or table for the field **TextField TheText).**

Example:

_CSVTEXT (1, 3, 5; "Error")

_CSVTEXTFORKEY

NEW

Returns the text from the specified source dannyhpri by searching by key.

_CSVTEXTFORKEY (DSNUM; KEYROW; KEYSTR; NVALUEROW; DEFSTR)

The arguments are:

DSNUM - number of the source data in the source list.

KEYROW - nomerstrokis key word for addressing a tabular representation of a CSV file (numbered from 1)

KEYSTR - keyword search.

NVALUEROW - nomerstroki with the desired value for the address in the table view a CSV file (numbered from 1)

DEFSTR - the default setting for the case of out-of-range address table, or if the data are not available.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value** or table for the field **TextField TheText).**

Example:

_ CSVTEXTFORKEY (1, 3; "yellow"; 5; "Error")

_CSVVALUE

NEW

Returns the integer value of the specified data source.

_CSVVALUE (DSNUM; NROW; NCOL; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NROW - nomerstroki to address in a table view CSV file (numbered from 1)

NCOL - nomerstolbtsa to address in a table view CSV file (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. Znacheniepredstavlyaetsya integer.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVVALUE (1, 1, 3, -1)

_CSVVALUED

NEW

Returns the value of the specified data source.

The return value can not be an integer.

_CSVVALUED (DSNUM; NROW; NCOL; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NROW - nomerstroki to address in a table view CSV file (numbered from 1)

NCOL - nomerstolbtsa to address in a table view CSV file (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value can not be an integer.

The value is in the right sections of one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVVALUE (1, 2, 2, -1.5)

_CSVVALUEDFORKEY

NEW

Returns the value of the specified source dannyhpri by searching by key.

The return value can not be an integer.

_CSVVALUEDFORKEY (DSNUM; KEYROW; KEYSTR; NVALUEROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

KEYROW - nomerstrokis key word for addressing a tabular representation of a CSV file (numbered from 1)

KEYSTR - keyword search.

NVALUEROW - nomerstroki with the desired value for the address in the table view a CSV file (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value can not be an integer.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVVALUEDFORKEY (1, 3; "blue"; 2, -1.5)

_CSVVALUEFORKEY

NEW

Vozvraschaettseloe value from the specified source dannyhpri by searching by key.

_CSVVALUEFORKEY (DSNUM; KEYROW; KEYSTR; NVALUEROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

KEYROW - nomerstrokis key word for addressing a tabular representation of a CSV file (numbered from 1)

KEYSTR - keyword search.

NVALUEROW - nomerstroki with the desired value for the address in the table view a CSV file (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value is an integer.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVVALUEFORKEY (1, 3; "black"; 1, -1)

_CSVVALUETYPE

NEW

Returns the type of data that are at the specified data source.

Possible return values - string, integer, floating-point number, color, date, the value is missing.

_CSVVALUETYPE (DSNUM; NROW; NCOL)

The arguments are:

DSNUM - number of the source data in the source list.

NROW - nomerstroki to address in a table view CSV file (numbered from 1)

NCOL - nomerstolbtsa to address in a table view CSV file (numbered from 1)

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_CSVVALUETYPE (1, 2, 6)

_CUT

_CUT (Arg; iarg)

Parameter **iarg** indicates how many digits after the decimal point remains. For negative **iarg** reset level before the decimal point.

Examples:

_CUT (123.4567; 3) = 123.456

_CUT (123.4567; -2) = 100

_CUT (123.4567; 0) = 123

_DATE

_DATE ()

Returns a string describing the last modified date in the document. The date format can vary for different systems and different countries.

Example:

_DATE () = 04.09.1999 (Mac)

_DATE () = 04 Sep 1999 (Win)

_DEG

_DEG (Arg)

Converts **arg** number from radians to degrees.

Examples:

_DEG (3.14) = 180

*_DEG (_PI () * 3) = 540*

_DOFORCONNECTED

NEW

_DOFORCONNECTED ("Nazvaniefunktsii"; id)

"Function Name" - the name of the function of the BASIC code objects.
id-idobekta in the document.

BASIC function calls the function with the appropriate name for objects that are attached to the object with the identifier given by the second parameter. In the attached objects should be compiled and run BASIC script.

Example:

*Add any object in the document properties of an object table table **Actions.** In the **Action** section write _DOFORCONNECTED ("AddText"; 9). After calling this function is executed CDBasic script function*

"AddText" those objects on the page, which it will be found and are attached to an object in a document with the identifier 9.

_ELLIPSE_ANGLE

_ELLIPSE_ANGLE (KoeffX; koeffY; iNumberGeometry; iNumberSegment)

Height * koeffY). The missing parameters for the construction of a segment taken from a number **iNumberSegment** in geometry at number **iNumberGeometry.**
This function is used as the default ones for a segment of an ellipse, in the column D.

_ELLIPSE_ASPECT

_ELLIPSE_ASPECT (KoeffX; koeffY; iNumberGeometry; iNumberSegment)
Returns the ratio of large to small radius in an ellipse centered at the point with local coordinates **(Width * koeffX; Height * koeffY).** The missing parameters for the construction of a segment taken from a numberi**NumberSegment** in geometry at number **iNumberGeometry.**
This function is used as the default ones for a segment of an ellipse, in column C.

_EVALTEXT

_EVALTEXT (Str)
Converts the string **str** into a number.

Examples:
_EVALTEXT ("123.456") = 123.456
_EVALTEXT ("123") = 123

_EXCELCOLORVALUE

NEW

Returns the color value written to the specified data source.

The color in the source data must be written in standard Web (eg # A180FF).

_EXCELCOLORVALUE (DSNUM; NSHEET; NROW; NCOL; DEFCOLOR)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered 1).

NROW - nomerstroki to address in a table view EXCEL files (numbered from 1)

NCOL - nomerstolbtsa to address in a table view EXCEL files (numbered from 1)

DEFCOLOR - the color value of the default output for the case of addressing the range of the table or if the data are not available.

The value is in the right sections of one or more tables of parameters of the object (for example in the table **FillFormat e** la **FillColor** field or in the table for the field **CustomProperties Value).**

Example:

_EXCELCOLORVALUE (3, 1, 5, 5; _RGB (255, 0, 0))

_EXCELGETCOLUMNFORKEY

NEW

Returns the number of columns in the specified source dannyhpri by searching by key.

_EXCELGETCOLUMNFORKEY (DSNUM; NSHEET; KEYROW; KEYSTR)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered 1).

KEYROW - nomerstroki to address in a table view EXCEL files (numbered from 1)

KEYSTR - a key string to search.

If the transmitted key string passed to the key column is not found, the function returns 0.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELGETCOLUMNFORKEY (3, 1, 2; "book")

_EXCELMAXELEMENT

NEW

Returns the maximum element of the string from the specified data source.

The return value is rounded to an integer.

Argumentamiyavlyayutsya:

_EXCELMAXELEMENT (DSNUM; NSHEET; NROW; DEFVAL)

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered 1).

NROW - nomerstroki to address in a table view EXCEL files (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value is an integer.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELMAXELEMENT (3, 1, 3, -1)

_EXCELMAXELEMENTD

NEW

Returns the maximum element of the string from the specified data source.

_EXCELMAXELEMENTD (DSNUM; NSHEET; NROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered 1).

NROW - nomerstroki to address in a table view EXCEL files (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data otsutstvuyut.Znachenie may not be an integer.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELMAXELEMENTD (3, 1, 1, -1.5)

_EXCELMINELEMENT

NEW

Returns the minimum element of the string from the specified data source.

The return value is rounded to an integer.

_EXCELMINELEMENT (DSNUM; NSHEET; NROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered 1).

NROW - nomerstroki to address in a table view EXCEL files (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value is an integer.

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELMINELEMENT (3, 1, 1, -1)

_EXCELMINELEMENTD

NEW

Returns the minimum element of the string from the specified data source.

_EXCELMINELEMENTD (DSNUM; NSHEET; NROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered 1).

NROW - nomerstroki to address in a table view EXCEL files (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data otsutstvuyut.Znachenie may not be an integer.

The value is in the right sections of one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELMINELEMENTD (3, 1, 2, -1)

_EXCELMINROWLENGTH

NEW

Returns the minimum length of a string (of all lines) for the specified data source.

_EXCELMINROWLENGTH (DSNUM; NSHEET)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered 1).

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELMINROWLENGTH(3;1)

_EXCELROWLENGTH

NEW

Returns the number of line items in the specified data source.

_EXCELROWLENGTH (DSNUM; NSHEET; NROW)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered 1).

NROW - nomerstroki to address in a table view EXCELfayla (numbered from 1)

The value is set to nuzhnyhsektsiyah one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELROWLENGTH (3, 1, 2)

_EXCELROWMAXELEMENT

NEW

Returns the maximum element of the string from the specified data source.

_EXCELROWMAXELEMENT (DSNUM; NSHEET; NROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered from 1)

NROW - the line number for the address in the table view EXCEL files (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value can not be an integer.

The value is in the right sections of one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELROWMAXELEMENT (3, 1, 4, -1.5)

_EXCELROWMINELEMENT

NEW

Returns the minimum element of the string from the specified data source.

_EXCELROWMINELEMENT (DSNUM; NSHEET; NROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered from 1)

NROW - the line number for the address in the table view EXCELfayla (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value can not be an integer.

The value is in the right sections of one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELROWMINELEMENT (3, 1, 1, -1)

_EXCELROWNUM

NEW

Returns the number of lines in the specified data source.

_EXCELROWNUM (DSNUM; NSHEET)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered from 1)

The value is in the right sections of one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELROWNUM (3, 1)

_EXCELTEXT

NEW

Returns the text written in the specified data source.

_EXCELTEXT (DSNUM; NSHEET; NROW; NCOL; DEFSTR)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered from 1)

NROW - the line number for the address in the table view EXCEL files (numbered from 1)

NCOL - Number column to address in a table view EXCEL files (numbered from 1)

DEFSTR-string is the default output for the case of addressing the range of the table or if the data are not available.

The value is in the right sections of one or more tables of parameters of the object (such as a table ***for the*** field ***CustomProperties Value*** or table for the field ***TextField TheText).***

Example:

_EXCELTEXT (3, 1, 2, 8; "Error")

_EXCELTEXTFORKEY

NEW

Returns the text from the specified source dannyhpri by searching by key.

_EXCELTEXTFORKEY (DSNUM; NSHEET; KEYROW; KEYSTR; NVALUEROW; DEFSTR)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered from 1)

KEYROW - Number strokis keyword to address in a table view EXCEL files (numbered from 1)

KEYSTR - keyword search.

NVALUEROW - the line number with the desired value for the address in the table view EXCEL files (numbered from 1)

DEFSTR - the default setting for the case of out-of-range address table, or if the data are not available.

The value is in the right sections of one or more tables of parameters of the object (such as a table for the field **CustomProperties Value** or table for the field **TextField TheText).**

Example:

_ EXCELTEXTFORKEY (4, 1, 4; "fix"; 2; "Error")

_EXCELVALUE

NEW

Returns the integer value of the specified data source.

_EXCELVALUE (DSNUM; NSHEET; NROW; NCOL; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered from 1)

NROW - the line number for the address in the table view EXCEL files (numbered from 1)

NCOL - Number column to address in a table view EXCEL files (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value is an integer.

The value is in the right sections of one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELVALUE (3, 1, 1, 3, -1)

_EXCELVALUED

NEW

Returns the value of the specified data source.

The return value can not be an integer.

_EXCELVALUED (DSNUM; NSHEET; NROW; NCOL; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered from 1)

NROW - the line number for the address in the table view EXCEL files (numbered from 1)

NCOL - Number column to address in a table view EXCEL files (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value can not be an integer.

The value is in the right sections of one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELVALUED (3, 1, 2, 2, -1.5)

_EXCELVALUEDFORKEY

NEW

Returns the value of the specified source dannyhpri by searching by key.

The return value can not be an integer.

_EXCELVALUEDFORKEY (DSNUM; NSHEET; KEYROW; KEYSTR; NVALUEROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered from 1)

KEYROW - Number strokis keyword to address in a table view EXCEL files (numbered from 1)

KEYSTR - keyword search.

NVALUEROW - the line number with the desired value for the address in the table view EXCEL files (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value can not be an integer.

The value is in the right sections of one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELVALUEDFORKEY (3, 1, 2; "enter"; 1, -1.5)

_EXCELVALUEFORKEY

NEW

Vozvraschaettseloe value from the specified source dannyhpri by searching by key.

_EXCELVALUEFORKEY (DSNUM; NSHEET; KEYROW; KEYSTR; NVALUEROW; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered from 1)

KEYROW - Number strokis keyword to address in a table view EXCEL files (numbered from 1)

KEYSTR - keyword search.

NVALUEROW - the line number with the desired value for the address in the table view EXCEL files (numbered from 1)

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value is an integer.

The value is in the right sections of one or more tables of parameters of the object (such as a table for the field **CustomProperties Value).**

Example:

_EXCELVALUEFORKEY (3, 1, 2; "bug"; 3, -1)

_EXCELVALUETYPE

NEW

Returns the type of data that are at the specified data source.

Possible return values - string, integer, floating-point number, color, date, the value is missing.

_EXCELVALUETYPE (DSNUM; NSHEET; NROW; NCOL)

The arguments are:

DSNUM - number of the source data in the source list.

NSHEET - number of bookmarks for the address in the table view EXCEL files (numbered from 1)

NROW - the line number for the address in the table view EXCEL files (numbered from 1)

NCOL - Number column to address in a table view EXCEL files (numbered from 1)

The value is in the right sections of one or more tables of parameters of the object (such as a table **for the** field **CustomProperties Value).**

Example:

_EXCELVALUETYPE (3, 1, 2, 6)

_FABS

_FABS (Arg)
If **arg** is nonzero, it returns the absolute value of **arg.**
If **arg** is zero, it returns 1.

_FABS (Str)
Returns the string **str** unchanged.

Examples:
_FABS (-3) = 3
_FABS (0) = 1
_FABS (1) = 1
_FABS ("Text") = "Text"

_FILENAME

_FILENAME ()
Returns the name of the file that stores the document.

Example:
_FILENAME () = "Chart.CDD"

_FILETEXT

NEW

Returns the text written in data source text file.

_FILETEXT (DSNUM; STARTPOS; SYMBOLCOUNT; DEFSTR)

The arguments are:

DSNUM - number of the source data in the source list.

STARTPOS - character position in a text file from which to start reading.

SYMBOLCOUNT - determines how many characters to read from a text file, starting spozitsii symbol defined STARTPOS. If the parameter SYMBOLCOUNT is 0, then read out the entire text of the position STARTPOS until the end of the file.

DEFSTR - the default setting for the case when the data are not available.

The value is in the right sections of one or more tables of parameters of the object (such as a table **for the** field **CustomProperties Value or table for the** field **TextField TheText).**

Example:

_ FILETEXT (2, 5, 20; "Error")

_FLOOR

_FLOOR (Arg)
Returns the largest integer not greater than **arg.**

Examples:
_FLOOR (123.4567) = 123
_FLOOR (-45.345) = -46
_FLOOR (0) = 0

_FULLFILENAME

_FULLFILENAME ()
Returns the name of the file that stores the document, with the full path.
Example:
_FULLFILENAME () = "D: \ ConceptDraw \ Chart.cdd" (Win)
_FULLFILENAME () = "MyDisk: DesktopFolder: Chart.cdd" (Mac)

_GETVALUE

Returns a string value from the **Value** Field **Data** Table parameters of the object.

_GETVALUE (NIND)

The argument is:

NIND - the line number (Field **Number)** Table **Data,** which in the **Value** field contains the value of interest.

The value is in the right sections of one or more tables of parameters of the object (such as a table **for the** field **CustomProperties Value).**

Example:

_GETVALUE (1)

_GETVALUEEL

Returns a string znachenieelementa list of the **Value** Field **Data** Table parameters of the object.

_GETVALUEEL (NIND; NNUM)

The arguments are:

NIND - the line number (Field **Number)** Table **Data,** which in the **Value** field contains the value of interest.

NNUM - Number list item row in the table in the **Data Value.**

The list is a set of values, separated by a comma.

The value is in the right sections of one or more tables of parameters of the object (such as a table **for the** field **CustomProperties Value).**

Example:

Example of a list: "9,777.777,999.99, ValueEl, 200"

_GETVALUEEL (2, 4)

As a result of the function in this case, we get the string "ValueEl"

_ GLUETOSERVICE

This feature ispolzuentsya to get coordinates of the center of rotation for the object to the bonding napravlyayushimliniiyam one of the control points of the object.

_GLUETOSERVICE (<Number of rail "," number of control points of the object>)

Depending on what kind of guide - vertical or horizontal - returns the corresponding coordinate.

If the rail is horizontal **GpinY** (coordinate along the axis Y), if it vertikaltnaya **GpinX** (coordinate along the axis X).

Notice. This feature is not designed for the user, the program itself is used for attaching an object to the guide lines.

_ GRADCOLOR

Sets the values of the parameters of the background color in **RGB.**

Syntax:

_GRADCOLOR (Color; percent).

color - tsvetvstandarte **RGB.**

percent - the percentage of the value of **"color".** From 0 to 100.

Examples.

_GRADCOLOR (_RGB (0, 255, 0); 50)

_ GRADCOLOR (FillColor; 10) -color is taken from the Fill Color field of the object parameters table

Use tables and sections FillFormat object parameters for the field FillPatColor.

_GRAVITY

_GRAVITY (Angle; limit1; limit2)
If **Angle limit1** more or less **limit2,** it returns 0
If **Angle** is in the interval **[limit1; limit2]** - returns the number **pi.**
The function is typically used to orient the text box so that the text at any position of the object was easy to read.

Examples:
_GRAVITY (30deg; 15 deg; 165 deg) = 0
_GRAVITY (195deg; 15 deg; 165 deg) = pi
_GRAVITY (Angle; -90 deg; 90 deg)

_ HTML2RGB

Converts a color from the standard color of the Web in the standard RGB.

Syntax:

_HTML2RGB ("Web color")

where the **"Web color"** - the color of the standard Web.

Example.

_HTML2RGB ("# 4C4C4C")

give gray.

Used for setting the colors in the parameter table (for parameters such as the section of the **FillColor Fill, LineColor LineProperties** section of the table object parameters.

_HYP

_HYP (X; Y)
Returns the length of the hypotenuse for a right triangle with legs of X and Y.

Example:
_HYP (4, 3) = 5

_IF

_IF (Arg1; arg2; arg3)
If the value of **arg1** - a non-zero number or non-empty string, the function returns **arg2,** otherwise - **arg3.**

Examples:
_IF (2> 1, 3, 4) = 3
_IF (""; 3, 4) = 4

_LG10

_LG10 (Arg)
Returns the logarithm of **arg**

_LN

_LN (Arg)
Returns the natural logarithm of **arg**

_LOCALX

_LOCALX (X; Y)
Translates a point (X; Y) coordinates from the global to local. Returns the X coordinate of the translated terms.

_LOCALY

_LOCALY (X; Y)
Translates a point (X; Y) coordinates from the global to local. Returns the Y coordinate of the translated terms.

_MAX

_MAX (Arg1; arg2)
Returns the larger of two numbers: **arg1** and **arg2.**

_MAX (Arg; str)

_MAX (Str; arg)

Returns the number **arg** (value of **str** are ignored).

_MAX (Str1; str2)

Returns the larger of the lengths of strings **str1** and **str2.**

Examples:
_MAX (4, 6) = 6
_MAX ("Text" '; "Big text") = 8
_MAX ("Text"; 7) = 7

_MEASURE

_MEASURE ()

Returns a string containing the abbreviated name of the current unit.

Example:
_MEASURE () = "Ft"

_MIN

_MIN (Arg1; arg2)

Returns the smaller of two numbers: **arg1** and **arg2.**

_MIN (Arg; str)

_MIN (Str; arg)

Returns the number **arg** (value of **str** are ignored).

_MIN (Str1; str2)

Returns the smaller of the lengths of strings **str1** and **str2.**

Examples:
_MIN (4, 6) = 6
_MIN ("Text" '; "Big text") = 8
_MIN ("Text"; 7) = 7

_MOD

_MOD (Arg1; arg2)

Returns the remainder after dividing by the number of **arg1 arg2**

_MOD (Str; arg)

_MOD (Arg; str)

Returns the number **arg,** if the other argument - the string **str.**

_MOD (Str1; str2)

Returns zero if the two arguments - the string.

Examples:
_MOD (19, 6) = 1
_MOD ("Text"; "Big text") = 0
_MOD ("Text"; 7) = 7

_NOT

_NOT (Arg)

If arg - zero or an empty string, it returns 1.

Otherwise it returns 0.

Examples:
_NOT (0) = 1
_NOT (123) = 0

_OR

_OR (Arg1; arg2)

Returns the bitwise "or";

_OR (Str1; str2)

Returns 1 - if at least one line - a non-empty, 0 - if both lines - empty.

_OR (Str; arg)
_OR (Arg; str)

Vozvraschaetchislo arg.

Examples:
_OR (1, 0) = 1
_OR ("Hello!"; "") = 1
_OR ("Text1"; "Text2") = 1
_OR ("Text"; 2) = 2

_PAGENUMBER

_PAGENUMBER ()

Returns the page number where the object belongs.

_PAGESCOUNT

_PAGESCOUNT ()

Returns the number of pages in the document.

_PAGEWIDTH

_PAGEWIDTH ()

Returns the width of your document. Note that the page size is set in the **Properties** dialog box **of the document,** the bookmark **page.**

_PI

_PI ()

Returns the value of pi

_POW

_POW (Arg1; arg2)

Returns the result of a number raised to the power **arg1 arg2.**

_POW (Str; arg)
_POW (Arg; str)

Returns the number **arg,** if the other argument - the string.

_POW (Str1; str2)
Returns zero if the two arguments - the string.

Examples:
_POW (2, 3) = 8
_POW ("Text"; "Big text") = 0
_POW ("Text"; 7) = 7

_RAD

_RAD (Arg)
Converts arg number from degrees to radians.

Examples
_RAD (90) = 1.57

_RAND

_RAND ()
Returns a random number between 0 and 32K.

_ RGB

Gives the color values of the parameters in RGB. Used to set colors (parameters such as the section of the **FillColor Fill, LineColor LineProperties** section of the table parameters of the object).

Syntax:

_RGB (R; G; B)

R, G, B - the components of red, green and blue, respectively. From 0 to 255.

Examples:

_RGB (255, 0, 0) gives a red color.

_ROUND

_ROUND (Arg; iarg)
Returns the result of rounding up the number of arg stochnostyu **iarg** digits after the decimal point.

Examples:
_ROUND (123.4567; 3) = 123.457
_ROUND (123.4567; -2) = 100
_ROUND (123.67; 0) = 124

_SCALE

_SCALE ()
Returns a string describing the scale of the current document in the form of "N: M"

Examples:
_SCALE () = "1: 1"
_SCALE () = "4 in: 1 ft"

_SETF

_SETF (Str; arg)
_SETF (Str; strarg)
The function is intended to change the values in table cells. In the string **str** is the name of the cell where to enter

the data. The **arg** parameter must contain a new numeric for the cell. **Strarg** parameter should contain a line with a new formula for the cell.

Examples:
_SETF (" Geometry1.X2";" Geometry2.X3 / 2 + Geometry3.X2 / 4")
_SETF (" Width"; 125 cm)

_SIGN

_SIGN (Arg)
Returns the sign of **arg:**
-1 If arg <0,
1 if arg> 0
0 if arg = 0

Examples:
_SIGN (123.4567) = 1
_SIGN (-123.4567) = -1
_SIGN (0) = 0

_SIN

_SIN (Arg)
Returns the sine of arg (it is in the range from -1 to 1).

_SINH

_SINH (Arg)
Returns the hyperbolic sine of **arg.**

_SMARTCONNECTORTEXTX

NEW!!!

It is used in the program to calculate the coordinates of the starting point of the text of the connector along the axis X.

Syntax:

_SMARTCONNECTORTEXTX (Width; Height); Event

There **Width** - the width, **Height** - the height, **Event** - an event.

Used in section **TextTransform** table parameters of the object for the field ***TextGPinX.***

Example:
_SMARTCONNECTORTEXTX (TextWidht; TextHeight); EventResize

_SMARTCONNECTORTEXTY

NEW!!!

It is used in the program to calculate the coordinates of the starting point of the text of the connector along the axis Y.

Syntax:

_SMARTCONNECTORTEXTY (Width; Height); Event

There **Width** - the width, **Height** - the height, **Event** - an event.

Used in section **TextTransform** table parameters of the object for the field ***TextGPinY.***

Example:
_SMARTCONNECTORTEXTY (TextWidht; TextHeight); EventResize

_SQRT

_SQRT (Arg)
Returns the square root of **arg.** For negative numbers, the value is not defined.

_ STYLED _ ENDSSIZE

Named to Fight this style returns the size of the arrows at the ends of lines.

Syntax:

_STYLED_ENDSSIZE ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program.
Returns a value from 0 to 4 for 5 possible sizes.
This function is used within a program object named in the appointment of style. In section **LineProperties** table parameters of the object for the field is set to ***LineEndsSize*** **_STYLED_ENDSSIZE ("style name"),** and the style name is always written in quotes.

_ STYLED _ FILLBGNDALPHA

NEW

Vozvraschaetvelichinu transparency of the background color of the fill for the named style.

Syntax:

_STYLED_FILLBGNDALPHA ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **FillFormat** table parameters of the object for the field is set
to **FillPatAlpha _STYLED_FILLBGNDALPHA ("Style Name").**

_ STYLED _ FILLCOLOR

Returns the fill color for the named style.

Syntax:

_STYLED_FILLCOLOR ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **FillFormat** table parameters of the object for the field is set
to **FillColor _STYLED_FILLCOLOR ("style name").**

_ STYLED _ FILLCOLORBGND

NEW

Returns the background color for the named style.

Syntax:

_STYLED_FILLCOLORBGND ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **FillFormat** table parameters of the object for the field is set
to **FillPatColor _STYLED_FILLCOLORBGND ("style name").**

_ STYLED _ FILLCOLORFGND

NEW

Returns the foreground color for the named style.

Syntax:

_STYLED_FILLCOLORFGND ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **FillFormat** table parameters of the object for the field is set
to **FillColor _STYLED_FILLCOLORFGND ("style name").**

_ STYLED _ FILLFGNDALPHA

NEW

Vozvraschaetvelichinu transparency of the foreground color for the named style.

Syntax:

_STYLED_FILLFGNDALPHA ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **FillFormat** table parameters of the object for the field is set
to **FillAlpha _STYLED_FILLFGNDALPHA ("style name").**

_ STYLED _ FILLPATCOLOR

Returns the background color of the fill for the named style.

Syntax:

_STYLED_FILLPATCOLOR ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **FillFormat table** parameters of the object for the field is set
to **FillPatColor _STYLED_FILLPATCOLOR ("style name").**

_ STYLED _ FILLPATTERN

Returns the palette for the named style.

_STYLED_FILLPATTERN ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.

In section **FillFormat table** parameters of the object for the field is set
to **FillPattern _STYLED_FILLPATTERN ("style name").**

_ STYLED _ FONTALPHA

NEW

Returns the transparency of the font color for the named style.

_STYLED_FONTALPHA ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **CharacterFormat** table parameters of the object for the field is set
to **Alpha _STYLED_FONTALPHA ("style name").**

_ STYLED _ FONTCHARLANG

Returns the number of languages for the named style.

_STYLED_FONTCHARLANG ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **CharacterFormat** table parameters of the object for the field **Language** is set
to **_STYLED_FONTCHARLANG ("style name").**

_ STYLED _ FONTCHARSET

NEW!!!

Returns for the named style.

_STYLED_FONTCHARSET ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.

_ STYLED _ FONTCOLOR

Returns the font color of a named style.

_STYLED_FONTCOLOR ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **CharacterFormat** table parameters of the object for the field is set
to **Color _STYLED_FONTCOLOR ("style name").**

_ STYLED _ FONTNUM

Returns the font number for the named style.

STYLED FONTNUM ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.

In section **CharacterFormat** table parameters of the object for the field is set to **Font _STYLED_ FONTNUM ("style name").**

_ STYLED _ FONTPOS

Returns the position of text characters (0 - plain text, 1 - superscript 2 - subscript) for the named style.

_STYLED_FONTPOS ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **CharacterFormat** table parameters of the object for the field is set to **Pos _STYLED_ FONTPOS ("style name").**

_ STYLED _ FONTSIZE

Returns the font size for the named style.

_STYLED_FONTSIZE ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **CharacterFormat** table parameters of the object for the field **Size** is set to **_STYLED_ FONTSIZE ("style name").**

_ STYLED _ FONTSPACING

Returns the distance between characters named for this style.

_STYLED_FONTSPACING ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **CharacterFormat** table parameters of the object for the field is set to **Spacing _STYLED_ FONTSPACING ("style name").**

_ STYLED _ FONTSTYLE

Returns a number that characterizes the set of styles for a block of text for the named style.

_STYLED_FONTSTYLE ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **CharacterFormat** table settings for a field object **Style** is set to **_STYLED_FONTSTYLE ("style name").**

_ STYLED _ LINEBEGIN

Returns the arrow type for the start of the geometry of an object to a named style.

_STYLED_LINEBEGIN ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **LineProperties** table parameters of the object for the field is set to **LineBegin _STYLED_LINEBEGIN ("style name").**

_ STYLED _ LINECOLOR

Returns the line color for the named style.

_STYLED_LINECOLOR ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **LineProperties** table parameters of the object for the field is set to **LineColor _STYLED_LINECOLOR ("style name").**

_ STYLED _ LINEEND

Returns the arrow type for the end of each geometry object for the named style.

_STYLED_LINEEND ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **LineProperties** table parameters of the object for the field is set to **LineEnd _STYLED_LINEEND ("style name").**

_ STYLED _ LINEPATTERN

Returns the property lines of discontinuity for the named style.

_STYLED_LINEPATTERN ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **LineProperties** table parameters of the object for the field is set to **LinePattern _STYLED_LINEPATTERN ("style name").**

_ STYLED _ LINEWEIGHT

Returns the line width for the named style.

_STYLED_LINEWEIGHT ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **LineProperties** table parameters of the object for the field is set to **LineWeight _STYLED_LINEWEIGHT ("style name").**

_ STYLED _ PARAAFTERINDENT

NEW

Returns the size of indenting a paragraph for that named style.

_STYLED_PARAAFTERINDENT ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In **Paragraph Format** section of the table object's parameters for the field is set to **RightInd _STYLED_PARAAFTERINDENT ("style name").**

_ STYLED _ PARAAFTERSPACING

Returns the interval between this and the following paragraph for that named style.

_STYLED_PARAAFTERSPACING ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In **Paragraph Format** section of the table object's parameters for the field is set to **AfterSpacing _STYLED_PARAAFTERSPACING ("style name").**

_ STYLED _ PARABEFOREINDENT

NEW

Returns the amount of space before the paragraph for that named style.

_STYLED_PARABEFOREINDENT ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In **Paragraph Format** section of the table object's parameters for the field is set to **LeftInd _STYLED_PARABEFOREINDENT ("style name").**

_ STYLED _ PARABEFORESPACING

Returns the interval between this and the preceding paragraph for that named style.

_STYLED_PARABEFORESPACING ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In **Paragraph Format** section of the table object's parameters for the field is set to **BeforeSpacing _STYLED_PARABEFORESPACING ("style name").**

_ STYLED _ PARABETWEENLINE

NEW

Returns the distance between lines of text for a range of named styles.

_STYLED_PARABETWEENLINE ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **ParagraphFormat** table parameters of the object for the field is set to **LineSpacing _STYLED_PARABETWEENLINE ("style name").**

_ STYLED _ PARAFIRSTLINE

Returns the size of the red line for the named style.

_STYLED_PARAFIRSTLINE ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.

In **Paragraph Format** section of the table object's parameters for the field is set to **FirstInd _STYLED_PARAFIRSTLINE ("style name").**

_ STYLED _ PARAHALIGNMENT

Returns a number describing the type of horizontal alignment of this section with respect to the text box named for this style.

_STYLED_PARAHALIGNMENT ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In **Paragraph Format** section of the table object's parameters for the field is set to **HAlign _STYLED_PARAHALIGNMENT ("style name").**

_ STYLED _ PARALEFTINDENT

Returns the size of the left indent for the paragraph style named.

_STYLED_PARALEFTINDENT ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In **Paragraph Format** section of the table object's parameters for the field is set to **LeftInd _STYLED_PARALEFTINDENT ("style name").**

_ STYLED _ PARALINESPACING

Returns the distance between lines of text for a range of named styles.

_STYLED_PARALINESPACING ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In **Paragraph Format** section of the table object's parameters for the field is set to **LineSpacing _STYLED_PARALINESPACING ("style name").**

_ STYLED _ PARARIGHTINDENT

Returns the size of the left indent for the paragraph style named for this.

_STYLED_PARARIGHTINDENT ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In **Paragraph Format** section of the table object's parameters for the field is set to **RightInd _STYLED_PARARIGHTINDENT ("style name").**

_STYLED_PENALPHA

NEW

Returns the value of the named prozrachnostiliniidlya style.

_STYLED_PENALPHA ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **LineProperties** table parameters of the object for the field is set to **LineAlpha _STYLED_ PENALPHA ("style name").**

_STYLED_PENCOLOR

NEW

Returns the line color for the named style.

_STYLED_PENCOLOR ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **LineProperties** table parameters of the object for the field is set to **LineColor _STYLED_PENCOLOR ("style name").**

_STYLED_PENPATTERN

NEW

Returns the property lines of discontinuity for the named style.

_STYLED_PENPATTERN ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **LineProperties** table parameters of the object for the field is set to **LinePattern _STYLED_PENPATTERN ("style name").**

_STYLED_PENWEIGHT

NEW

Returns the line width for the named style.

_STYLED_PENWEIGHT ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **LineProperties** table parameters of the object for the field is set to **LineWeight _STYLED_PENWEIGHT ("style name").**

_ STYLED _ SHADOWBGNDALPHA

NEW

Returns the transparency plan tsvetazadnego shadow of a named style.

_STYLED_SHADOWBGNDALPHA ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.

In section **FillFormat table** parameters of the object for the field is set to **ShadowPatAlpha _STYLED_SHADOWBGNDALPHA ("style name").**

_STYLED_SHADOWCOLOR

Returns the foreground color for the shade of a named style.

_STYLED_SHADOWCOLOR ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **FillFormat table** parameters of the object for the field is set to **ShadowColor _STYLED_SHADOWCOLOR ("style name").**

_ STYLED _ SHADOWCOLORBGND

NEW

Returns the background color for the shadow of a named style.

_STYLED_SHADOWCOLORBGND ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **FillFormat table** parameters of the object for the field is set to **ShadowPatColor _STYLED_SHADOWCOLORBGND ("style name").**

_ STYLED _ SHADOWCOLORFGND

NEW

Returns the foreground color for the shade of a named style.

_STYLED_SHADOWCOLORFGND ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **FillFormat table** parameters of the object for the field is set to **ShadowColor _STYLED_SHADOWCOLORFGND ("style name").**

_ STYLED _ SHADOWFGNDALPHA

NEW

Returns the transparency of the foreground shadow of a named style.

_STYLED_SHADOWFGNDALPHA ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **FillFormat table** parameters of the object for the field is set to **ShadowAlpha _STYLED_SHADOWFGNDALPHA ("style name").**

_ STYLED _ SHADOWPATCOLOR

Returns the background color of the shadow of the figure for the named style.

_STYLED_SHADOWPATCOLOR ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **FillFormat table** parameters of the object for the field is set
to **ShadowPatColor _STYLED_SHADOWPATCOLOR ("style name").**

_ STYLED _ SHADOWPATTERN

Returns the fill pattern for the shade of a named style.

_STYLED_SHADOWPATTERN ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **FillFormat table** parameters of the object for the field is set
to **ShadowPattern _STYLED_SHADOWPATTERN ("style name").**

_ STYLED _ TXTBKGNDCOLOR

Returns the background color, which displays the text for the named style.

_STYLED_TXTBKGNDCOLOR ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **TextBlockFormat** table parameters of the object for the field is set
to **TextBkgnd _STYLED_TXTBKGNDCOLOR ("style name").**

_ STYLED _ TXTBOTTOMMARGIN

Returns the indentation from the bottom of the text box named for the style.

_STYLED_TXTBOTTOMMARGIN ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **TextBlockFormat** table parameters of the object for the field is set
to **BottomMargin _STYLED_TXTBOTTOMMARGIN ("style name").**

_ STYLED _ TXTDEFTABSTOP

NEW

Returns the tab in a text box named for this style.

_STYLED_TXTDEFTABSTOP ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **TextTabsTable** table parameters of the object for the field is set
to **DefaultTabStop _STYLED_TXTDEFTABSTOP ("style name").**

_ STYLED _ TXTLEFTMARGIN

Returns the indentation from the left border of the text frame to a named style.

_STYLED_TXTLEFTMARGIN ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **TextBlockFormat** table parameters of the object for the field is set to **LeftMargin _STYLED_TXTLEFTMARGIN ("style name").**

_ STYLED _ TXTRIGHTMARGIN

Gets the indentation on the right edge of the text box named for the style.

_STYLED_TXTRIGHTMARGIN ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **TextBlockFormat** table parameters of the object for the field is set to **RightMargin _STYLED_TXTRIGHTMARGIN ("style name").**

_ STYLED _ TXTTOPMARGIN

Returns the indentation of the upper limit for the text box named style.

_STYLED_TXTTOPMARGIN ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **TextBlockFormat** table parameters of the object for the field is set to **TopMargin _STYLED_TXTTOPMARGIN ("style name").**

_ STYLED _ TXTVALIGN

Returns an integer representing the type of vertical alignment of text relative to the text box named for this style.

_STYLED_TXTVALIGN ("Style Name")

The argument is the name of an existing named style that is created by the user or is already in the program. This function is used within a program object named in the appointment of style.
In section **TextBlockFormat** table parameters of the object for the field is set to **VAlign _STYLED_TXTVALIGN ("style name").**

_TAN

_TAN (Arg)
Returns the tangent of **arg.**

_TANH

_TANH (Arg)
Returns the hyperbolic tangent of **arg.**

_TEXTHEIGHT

_TEXTHEIGHT (Str; arg)
The function is intended to clarify what would be the height of the text block in appointing him to the width of the **arg.** As the string **str** is commonly used contents of a text field object (cell **TheText).** When calculating the height of the text takes into account the current configuration of the object (styles, indentation of paragraphs, etc.).

Examples:
_TEXTHEIGHT (TheText; Width)
_TEXTHEIGHT (TheText; 2 in)

_TEXTLEFT

_TEXTLEFT (Str; iarg)
Returns a substring of **str,** consisting of the first **iarg** characters (left substring).

Example:
_TEXTLEFT ("A big text."; 5) = "A big"

_TEXTLENGTH

_TEXTLENGTH (Str)
Returns the length of the string **str** (number of characters per line).

Example:
_TEXTLENGTH ("A big text.") = 11

_TEXTRIGHT

_TEXTRIGHT (Str; iarg)
Returns a substring of **str,** consisting of the last **iarg** characters (right substring).

Example:
_TEXTRIGHT ("A big text"; 4) = "text"

_TEXTWIDTH

_TEXTWIDTH (Str)
Returns the width of the string **str** according to the current text object settings (styles, indentation of paragraphs, etc.). Typically, the function is used to assign the width of the text frame is equal to the longest string in the text object.

Example:
_TEXTWIDTH (TheText)

_TIME

_TIME ()
Returns the last time changes to the document in the format "hours: minutes: seconds."

Example:
_TIME () = "19:27:13"

_VALTOTEXT

_VALTOTEXT (Arg)
Converts **arg** number to a string and returns it.

Example:
_VALTOTEXT (567.89) = "567.89"

_VALTOTEXTMES

_VALTOTEXTMES (Arg)
Converts number to string arg in view of current units of measurement specified in the document.

Examples:

_VALTOTEXTMES (15) = "1/16"

_VALTOTEXTMES (1.5 in) + "in." = "1 1/2 in."

_WORLDX

_WORLDX (X; Y)

Translates a point (X; Y) from local to global coordinates. Returns the X coordinate of the translated terms.

_WORLDY

_WORLDY (X; Y)

Translates a point (X; Y) from local to global coordinates. Returns the Y coordinate of the translated terms.

_XOR

_XOR (Arg1; arg2)

Returns an exclusive "or";

_XOR (Str1; str2)

Returns 1 - if the one and only one line - a non-empty, 0 - if both lines - empty, or both - non-empty.

_XOR (Str; arg)

_XOR (Arg; str)

Returns the number **arg.**

Examples:

_XOR (1, 1) = 0

_XOR ("Text1"; "Text2") = 0

_XOR ("Text"; 2) = 2

_XPATHVALUE

NEW

Returns the integer value of the specified XMLfaylaistochnika data.

_XPATHVALUE (DSNUM; XPATHEXPR; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

XPATHEXPR - XPATHvyrazhenie.

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available. The value is an integer.

The value is in the right sections of one or more tables of parameters of the object (such as a table **for the** field **CustomProperties Value).**

Example:

_ XPATHVALUE (4; "/ Localization / XPATHValue"; -1)

_XPATHVALUED

NEW

Returns the value of the specified XMLfaylaistochnika data.

The return value can not be an integer.

_XPATHVALUED (DSNUM; XPATHEXPR; DEFVAL)

The arguments are:

DSNUM - number of the source data in the source list.

XPATHEXPR - XPATHvyrazhenie.

DEFVAL - the default output for the case of addressing the range of the table or if the data are not available.

The value is in the right sections of one or more tables of parameters of the object (such as a table **for the** field **CustomProperties Value).**

Example:

_ XPATHVALUED (4; "/ Localization / XPATHValueD"; -1.5)

_XPATHTEXT

NEW

Returns the text written in ukazannomXMLfayleistochnika data.

_XPATHTEXT (DSNUM; XPATHEXPR; DEFSTR)

The arguments are:

DSNUM - number of the source data in the source list.

XPATHEXPR - XPATHvyrazhenie.

DEFSTR - the default setting for the case when the data are not available.

The value is in the right sections of one or more tables of parameters of the object (such as a table **for the** field **CustomProperties Value or table for the** field **TextField TheText).**

Example:

_XPATHTEXT (4; "/ Localization / XPATHText"; Error ")

Appendix 2. CDBasic reference

ConceptDraw Basic Reference

Welcome to ConceptDraw Basic Reference. The Reference gives you fast access to information about all the ConceptDraw Basic language elements: statements, operators, constants, error messages, objects, built-in methods and functions, and more.

The Reference contains the following sections:

- Overview
- Conceptual Information
- Language Core Reference
- Objects Reference
- Trappable errors
- Glossary

Overview

Overview

ConceptDraw Basic is a high-level scipting language. Starting from version 5.0 ConceptDraw introduces support for its propietary built-in scripting language - ConceptDraw Basic. This adds the following advantages:

- Extends the functionality of ConceptDraw according to the needs of the users.
- Allows to process and visualize external data in ConceptDraw.
- Makes possible integration of third-Elementy application with ConceptDraw.
- Enables a wide range of cross-platform solutions, based on ConceptDraw.

ConceptDraw Basic technology (unlike Automation on Windows, AppleScript on the Macintosh) is fully cross-platform , working in the ConceptDraw environment. The built-in scripting language realizes the specification of the modern high-level scripting language combined with support for ConceptDraw objects and database access objects. The supported list of ConceptDraw objects provides virtually unlimited control over documents, application windows, libraries, pages and shapes.

ConceptDraw Basic has the power and simplicity of modern realizations of the BASIC language. The language core of ConceptDraw Basic is almost fully compatible with such popular realizations of BASIC, such as Visual Basic, REALbasic.

With the introduction of ConceptDraw Basic technology ConceptDraw becomes one of the most powerful platforms for your custom visual solutions.

Conceptual Information

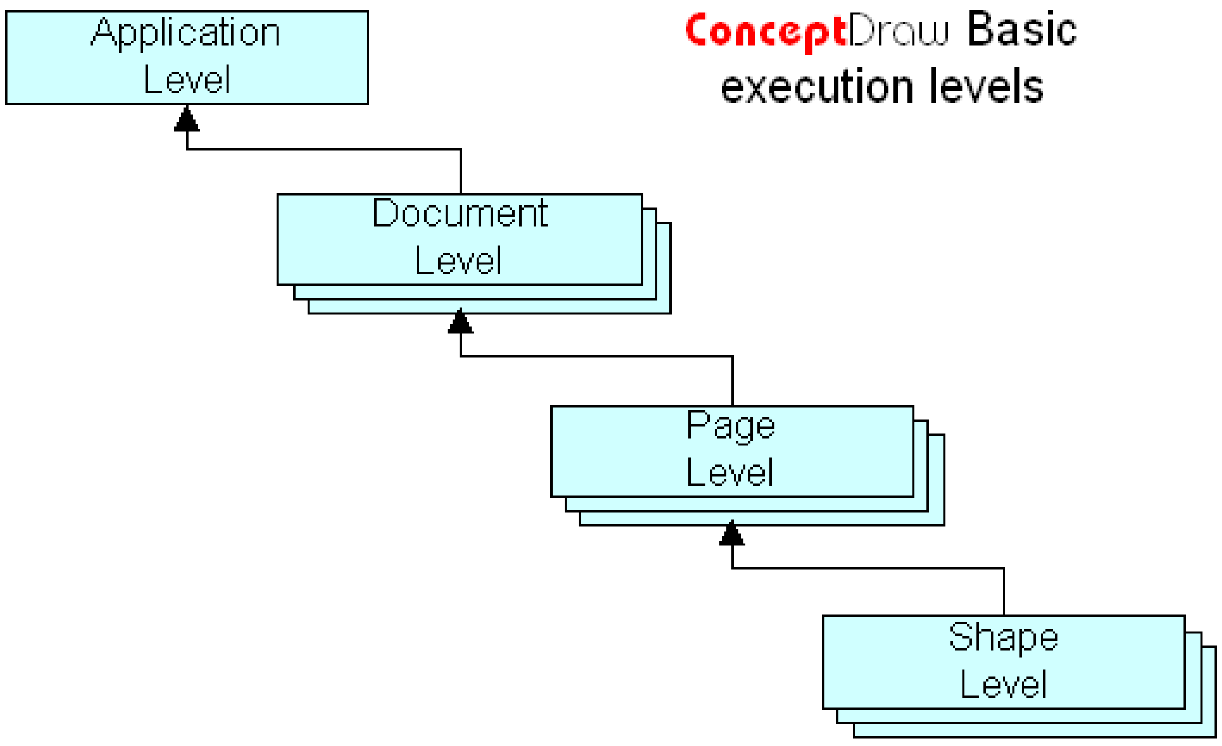
This section describes the structure and principles of writing scripts in ConceptDraw Basic. It contains the following paragraphs:

- Execution Levels
- Storing Scripts
- Editing Scripts
- The Structure of a Script
- Compilation and Execution of Scripts

Execution levels.

ConceptDraw supports four execution levels of the ConceptDraw Basic scripting language: Application level, Document level, Page level, Shape level. This means that for any ConceptDraw document or it's page or any shape in the document you can assign a program written in ConceptDraw Basic. Also one can create a program on the entire application level. Any execution level contains at least a built-in module with program code in ConceptDraw Basic.

Execution levels of ConceptDraw Basic are organized in a hierarchy (see the figure below), which reflects how global variables and procedures are inherited from higher levels to the lower ones.



This means, that a script, created for any shape (on the shape level) also shows all global variables and procedures that belong to higher levels: Page, Document and Application. In its turn, a page-level script shows all global variables and procedures of the Document and Application levels. And finally, a document-level script shows global variables and procedures of the Application level. Thus, the hierarchy of execution levels determines the functional purpose of ConceptDraw Basic scripts at different levels.

Application levels script is intended for re-assigning the behavior of the entire application, and also for defining global variables and procedures, which may be often used in various documents. For instance, with the help of interface configuration and an application-level script in ConceptDraw Basic it's possible to turn ConceptDraw into a specialized application for computer network designers. One should just write the commonly used routines (for instance, calculation of the cost of the components) as application-level scripts and run them using the user-defined menu. Then the user will be able to automatically calculate the cost of the components for any network diagram.

Document level script is intended for document-specific calculations and also for defining global variables and procedures, used in the code of different pages or shapes of the document. For instance, a document-level script can be used to define specific procedures for creating templates. This may look like a wizard, that asks questions specific to a certain document type. Based on the user input, the script can determine the number and size of pages, create these pages and place necessary shapes on them.

Page level script is intended for calculations and actions, specific to a certain page of the document, as well as for defining global variables and procedures, used in the code of the shapes on that page. Scripts at this level may be used together with document-level scripts when creating templates. Creating graphic objects (shapes) is slightly easier at the page level, than at the document level.

Shape level script is intended for calculations, specific to certain graphic object (shape). For instance, it allows to program an element of a bar chart in such a way, that it can reflect values from a data base or an external file. Library shapes can also have scripts.

Storing Scripts

The code of the scripts of document, page and shape levels is stored together with the object, to which the script is assigned. For instance, scripts for the document and its pages and shapes are stored within the document. For shapes in a library the code is stored with the library.

An application-level script is stored in a file with reserved name "AppCDBasicScript.cdb", located in the application data folder. For example, full path to an external module of the application-level script on the Windows platform may look like this: "C:\Documents and Settings\Dime1.DIME\Application Data\CSOdessa\ConceptDraw\AppCDBasicScript.cdb". An application-level script is only saved if compilation was successful.

ConceptDraw Basic allows to use external modules with ConceptDraw Basic code by means of the inline command **#Include**. This lets create various external libraries of routines.

Source code of ConceptDraw Basic scripts is stored as text in the UTF-8 encoding, allowing to use string constants and comments in any language.

Editing Scripts

For editing and debugging scripts ConceptDraw has a built-in ConceptDraw Basic script editor. This editor allows to edit scripts of all execution levels, as well as external modules, connected by the **#Include** command. Besides, ConceptDraw Basic script editor lets compile and run scripting programs at available execution level. The "CDBasic Output" window serves for debugging and showing warnings and errors.

To edit external modules you can use any other text editor. However, if the code contains comments or string constant, that include national characters (non-ANSI symbols), the editor should be able to save text in the UTF-8 encoding.

The Structure of a Script

A script at any execution level contains the global execution area, and a set of user procedures, defining local execution areas.

In the global area global variables are defined, user procedures are declared and defined, external procedures declared. Also in the global area is located the code, executed immediately at launch. Variables and named constants, defined in the global area, can be visible in all user procedures, defined lower in the code from where they were declared.

Local execution areas contain user procedures. Definitions of user procedures start with the statements Sub or Function, and end with End Sub or End Function respectively. Variables, defined in a local area, are visible within this area only. This allows to use local variables and named constants with same names in different procedures.

Any variables is visible down the code from where it was declared until the end of its visible area.

Below is an example of a ConceptDraw Basic script:

```
Dim gData(256) As Double    ' Declare global variable gData as Double array
Dim gCount As Long    ' Declare global variable gCount as Long
' Definition of InitGlobalData() procedure
Sub InitGlobalData()    ' procedure begin
        ' Make global data initialization
        For i = 0 To 256
              gData(i)=i
        Next
End Sub    ' procedure end
' Definition of TraceGlobalData() procedure
Sub TraceGlobalData ()    ' procedure begin
        For i = 0 To 256
        Trace gData(i)
        Next
End Sub    ' procedure end
' Definition of RecalcGlobalData() procedure
Sub RecalcGlobalData ()    ' procedure begin
        For i = 0 To 256
              ' Do some calculation here
              gData(i)=gData(i)+Rnd()
        Next
End Sub    ' procedure end
gCount = 0    ' set gCount to 0
```

```
InitGlobalData() ' Call procedure for global data initialization
Stop
```

Compilation and Execution of Scripts

Scripts are executed by the built-in virtual machine of ConceptDraw Basic. The source code in ConceptDraw Basic is first compiled into so called p-code of the virtual machine, which is then executed. So, the life cycle of a program in ConceptDraw Basic can be divided into two stages - compilation and execution.

During compilation the compiler finds all syntactic errors and informs about them in the "CDBasic Output" window. Normally (where possible) it displays the error number, short error description and shows the source module and the line number, in which the error was found.
When compilation of a script starts, the scripts of higher execution level are compiled automatically if they weren't compiled earlier. When writing scripts you should remember that namespaces of variables and procedures at different levels should not overlap. If variables or procedures were earlier defined at a higher execution level, this will lead to a compilation error of "Duplicate definition" type. Also, a compilation error will be caused by declaring variables or constants with names, coinciding with the names of the built-in constants or run-time procedures. The same would happen with reserved words of the ConceptDraw Basic language. Detailed description of compilation errors can be found in the "**Trappable errors**" section.

Successfully compiled code of a ConceptDraw Basic script can be executed. It can be launched either by the user from the menu or a toolbar button, or automatically when loading the script-containing object.
Once a script is launched, scripts of the upper execution levels are launched automatically if they haven't been launched by the moment (not resident).
ConceptDraw Basic starts running the script from executing the statements of the global area. Procedures are skipped at this stage, because procedures start executed only when they are called. Once the statements of the global area have been executed, or on executing the Stop statement, the program goes to the stand-by mode, remaining resident. In this case any procedure can be called from scripts of lower execution level, or from the procedures that process reserved events. For instance, a document-level script can add items to the custom menu of the document and process them by using its own procedures. Below is an example of such program:

```
' Definition of procedure
Sub MenuItem1_CmdProc(cmdArgs As String)
       Trace "MenuItem1 : " & cmdArgs
        ' ...
        ' ...
End Sub
Dim mi As MenuItem
' Enable Document custom menu
thisDoc.CustomMenu.Caption = "My Doc menu"
' Add menu item
set mi = thisDoc.CustomMenu.AddMenuItem(0)
' Set menu item caption
mi.Caption = "Item 1"
mi.OnCmdArgs = "Args string from menu item"
```

```
' Set processing procedure
mi.SetCmdProcessing("MenuItem1_CmdProc")
' Suspends execution
Stop
```

On executing the End statement the program stops. All global variables are cleared, and all procedures defined at this level become inaccessible for subsequent calls.

In automatic mode a script is launched as soon as the object containing it is loaded. That is, an application-level script is run as soon as the application is launched. After you open a document or a template, a document-level script is launched. Then, if the document-level script remains resident, the scripts at all page levels are executed subsequently, starting from the first page. Once a page-level program has been executed, and provided it remains resident (i.e. it wasn't stopped by the End statement), scripts of the shapes on the page are launched. A shape-level script is also started automatically, once the script-containing object has been inserted into the document from a library or duplicated.

A flag in the application preferences dialog controls whether scripts may be launched automatically or not.

Language Core Reference

Language Core Reference

- Statements
- Operators
- Functions
- Constants
- Keywords
- Data Type Summary

Abs Function

Abs Function

Returns a value of the same type that is passed to it specifying the absolute value of a number.

Syntax

Abs([*num*])

The optional *num* argument is any valid numeric expression. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 0.

Remarks

The absolute value of a number is its unsigned magnitude. For example, **Abs**(-1) and **Abs**(1) both return 1.

Example

```
Dim MyNumber
MyNumber = Abs(36.6)   ' Returns 36.6.
MyNumber = Abs(-36.6)  ' Returns 36.6.
```

See Also [Sgn Function]

ADDRESSOF Operator

ADDRESSOF Operator

A unary operator that returns the address of a variable.

Syntax

result = AddressOf *varname*

The AddressOf operator syntax has these Elements:

Element	**Description**
result	Required; any numeric variable.
varname	Required; any variable.

Remarks

The **AddressOf** operator returnns the address of any variable. If the variable was declared as object and wasn't initialized, **AddressOf** returns 0. If the variable was declared and initialized with the Set statement, **AddressOf** returns the address of the variable in memory.

Example

```
Dim MyAddress, AddressOfMyAddress, MyPoint as DPoint
MyAddress = AddressOF MyPoint    ' Returns 0.
trace MyAddress
Set MyPoint = New DPoint
MyAddress = AddressOF MyPoint    ' Returns address of object MyPoint.
trace Hex(MyAddress)
AddressOfMyAddress = AddressOF MyAddress ' Returns address of variable
MyAddress.
trace Hex(AddressOfMyAddress)
```

See Also [Operators]

+ Operator

+ Operator

Used to sum two numbers.

Syntax

result = expression1 + expression2

The + operator syntax has these Elements:

Element	**Description**
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If at least one expression is not a Variant, the following rules apply:

If	**Then *result* is**
Both expressions are numeric data types (**Byte**, **Boolean**, **Integer**, **Long**, **Single**, **Double**, **Date**)	Add.
Both expressions are **String**	Concatenate.
One expression is a numeric data type and the other is any **Variant** except **Null**	Add.
One expression is a **String** and the other is any **Variant** except **Null**	Concatenate.

If both expressions are Variant expressions, the following rules apply:

If	**Then**
Both **Variant** expressions are numeric	Add.
Both **Variant** expressions are strings	Concatenate.

One **Variant** expression is numeric and the other is a string	Add.

For simple arithmetic addition involving only expressions of numeric data types, the data type of *result* is usually the same as that of the most precise expression. The order of precision, from least to most precise, is **Byte**, **Integer**, **Long**, **Single**, **Double**. The following are exceptions to this order:

If	**Then *result* is**
The data type of result is a **Long**, **Single**, or **Date** variant that overflows its legal range,	converted to a **Double** variant.
The data type of result is a **Byte** variant that overflows its legal range,	converted to an **Integer** variant.
The data type of result is an **Integer** variant that overflows its legal range,	converted to a **Long** variant.
A **Date** is added to any data type,	a **Date**.

Example

```
Dim MyNumber, Var1, Var2
MyNumber = 2 + 2 ' Returns 4.
trace MyNumber
MyNumber = 4257.04 + 98112 ' Returns 102369.04.
trace MyNumber
Var1 = "34": Var2 = 6 ' Initialize mixed variables.
MyNumber = Var1 + Var2 ' Returns 40.
trace MyNumber
Var1 = "34": Var2 = "6" ' Initialize variables with strings.
MyNumber = Var1 + Var2 ' Returns "346" (string concatenation).
trace MyNumber
```

See Also Operators

AND Operator

AND Operator

Used to perform a logical conjunction on two expressions.

Syntax

result = *expression1* **And** *expression2*

The And operator syntax has these Elements:

Element	**Description**
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If both expressions evaluate to **True**, result is **True**. If either expression evaluates to **False**, result is **False**. The following table illustrates how result is determined:

If expression1 is	**And expression2 is**	**The result is**
True	True	True
True	False	False
False	True	False
False	False	False

The And operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following table:

If expression1 is	**And expression2 is**	**The result is**
0	0	0
0	1	0
1	0	0
1	1	1

Example

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null   ' Initialize variables.
MyCheck = A > B And B > C   ' Returns True.
trace MyCheck
MyCheck = B > A And B > C   ' Returns False.
trace MyCheck
MyCheck = A > B And B > D   ' Returns True.
trace MyCheck
MyCheck = A And B   ' Returns 8 (bitwise comparison).
trace MyCheck
```

See Also [Operators]

Asc Function

Asc Function

Returns an Integer representing the character code corresponding to the first letter in a string.

Syntax

Asc([*string*])

The optional *string* argument is any valid string expression. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 0.

Remarks

The range for returns is 0 – 255 on non-DBCS systems, but –32768 – 32767 on DBCS systems.

Example

```
Dim MyNumber
MyNumber = Asc("A")    ' Returns 65.
MyNumber = Asc("a")    ' Returns 97.
MyNumber = Asc("Apple")    ' Returns 65.
```

See Also [Chr Function], [Type Conversion Functions]

Atn Function

Atn Function

Returns a **Double** specifying the arctangent of a number.

Syntax

Atn([*num*])

The optional *num* argument is a **Double** or any valid numeric expression. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 0.

Remarks

The **Atn** function takes the ratio of two sides of a right triangle (*num*) and returns the corresponding angle in radians. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

The range of the result is -pi/2 to pi/2 radians.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

> **Note Atn** is the inverse trigonometric function of **Tan**, which takes an angle as its argument and returns the ratio of two sides of a right triangle. Do not confuse **Atn** with the cotangent, which is the simple inverse of a tangent (1/tangent).

Example

```
Dim pi
pi = 4 * Atn(1)    ' Calculate the value of pi.
```

See Also **Cos Function**, **Sin Function**, **Tan Function**

Beep Statement

Beep Statement

Plays a sound signal through computer's built-in speaker.

Syntax

Beep

Remarks

Frequency and lenght of the sound signal depends on computer hardware and software and vary with different computers. .

Example

In this example the **Beep** statement is used to play three sound signals through the speaker.

```
Dim I
For I = 1 To 3  ' The cycle repeats 3 times
 Beep    '  Play sound signal
Next I
```

See Also

Bin Function

Bin Function

Returns a **FixStr** (**String**) value representing the binary value of a number.

Syntax

Bin[$]([*number*])

The optional *number* argument is any valid numeric expression or string expression in the range from -2147483648 to 2147483647. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 0.

Remarks

If *number* is not already a whole number, it's rounded to the nearest whole number before being evaluated. If *number* is **Empty** or **Null**, the function returns 0. For any other number the **Bin** function returns up to 32 binary symbols.

You can represent binary numbers directly by preceding numbers in the proper range with &B. For example, &B10 represents decimal 2 in binary notation.

The **Bin$** returns **String** values. The **Bin** form returns **FixStr** values.

Example

```
Dim MyBin
MyBin = Bin(5)   ' Returns 101.
MyBin = Bin(10)   ' Returns 1010.
MyBin = Bin(459)   ' Returns 111001011.
```

See Also Oct Function,Hex Function,Type Conversion Functions

Call Statement

Call Statement

Transfers control to a **Sub** procedure, **Function** procedure, or dynamic-link library (DLL) procedure.

Syntax

[**Call**] *name* ([*argumentlist*])

The **Call** statement syntax has these Elements:

Element	Description
Call	Optional; keyword. Supported for compatibility with other versions of BASIC.
name	Required. Name of the procedure to call.
argumentlist	Optional. Comma-delimited list of variables, array items, or expressions to pass to the procedure. Components of *argumentlist* may include the keywords **ByVal** or **ByRef** to describe how the arguments are treated by the called procedure.

Remarks

You are not required to use the **Call** keyword when calling a procedure. However, if you use the **Call** keyword to call a procedure that requires arguments, *argumentlist* must be enclosed in parentheses. If you use either **Call** syntax to call any intrinsic or user-defined function, the function's return value is discarded.

Example

This example illustrates how the **Call** statement is used to transfer control to a **Sub** procedure, an intrinsic function.

```
Declare Sub PrintToOutputWindow(AnyString As String)
' Call a Sub procedure.
Call PrintToOutputWindow("Hello World")
' The above statement causes control to be transfered to the following
' Sub procedure.
Sub PrintToOutputWindow(AnyString As String)
```

```
   Trace AnyString   ' Print to the Output window.
End Sub
' Call is an intrinsic function. The returned value of the function is
discarded.
Call MsgBox("Call an intrinsic MsgBox function")
```

See Also Declare Statement , Function Statement , Sub Statement

Type Conversion Functions

Type Conversion Functions

Each function coerces an expression to a specific data type.

Syntax

CBool([*expression*])

CByte([*expression*])

CDbl([*expression*])
CInt([*expression*])
CLng([*expression*])
CSng([*expression*])
CVar([*expression*])
CStr([*expression*])
CDate([*expression*])

CVDate([*expression*])

The optional *expression* argument is any string expression or numeric expression.

Return Types

Function	Return Type
CBool	Bool
CByte	Byte
CDbl	Double
CInt	Integer

CLng	**Long**
CSng	**Single**
CVar	**Variant**
CStr	**String**
CDate	**Date**
CVDate	**Date**

Remarks

If the *expression* passed to the function is outside the range of the data type being converted to, it's transformed according to the following rules (on example of **CInt**):

CInt

-32768(min).....................0........................32767(max)

CInt(32768) returns -32768

CInt(32769) returns -32767

...

CInt(-32769) returns 32767

CInt(-32770) returns 32766

In general, you can document your code using the data-type conversion functions to show that the result of some operation should be expressed as a Elementicular data type rather than the default data type.

When the fractional Element is exactly 0.5, **CInt** and **CLng** as well as **CByte** always round it to the nearest even number. For example, 0.5 rounds to 0, and 1.5 rounds to 2. **CInt** and **CLng** differ from the **Fix** and **Int** functions, which truncate, rather than round, the fractional Element of a number. Also, **Fix** and **Int** always return a value of the same type as is passed in.

If the *expression* argument is omitted, **CVar** and **CStr** return an empty string, other functions return 0.

A **CVDate** function is identical to **CDate** and is provided for compatibility with other versions of BASIC.

Example

```
Dim  MyDouble, MyInteger, MyDate
MyDouble = CDbl("3.2")    ' Convert result to a Double -> 3.2
MyInteger = CInt(MyDouble)    ' Convert result to a Integer -> 3
MyDate = CDate(MyInteger)    ' Convert result to a Date -> 2 Jan 1990
```

See Also Fix Function, Int Function , Round Function

ChDir Statement

ChDir Statement

Sets a new current directory or folder.

Syntax

ChDir *path*

The required argument *path* is a string that specifies a new current directory (or folder). The *path* argument may contain the disk name. If the disk name is not specified, **ChDir** assumes it's the current disk.

Remarks

The **ChDir** statement changes the current directory, but doesn't change the current disk. For instance, if drive C is current, the command below will change the current directory to one on drive D, however drive C remains the current drive:

ChDir "D:\TMP"

ChDir Statement (Apple Power Macintosh)

On Power Macintosh the current disk is always changed to the disk, specified in the path. A full path should start with the volume name, a relative path starts with a colon (:). ChDir allows using random names in the path line. ChDir "MacDrive:Tmp" ' on the Macintosh.

Note, that Microsoft Windows and Macintosh use different symbols for relative path changes:

ChDir ".." ' Go up one level in Microsoft Windows.
ChDir "::" ' Go up one level on the Macintosh.

Example

In this example the ChDir statement is used to change the current directory or folder.

```
' Change current directory or folder to "MYDIR".
ChDir "MYDIR"
'In Microsoft Windows:
' Current disk is drive "C:". The following statement sets a new current
' directory on drive "D:". "C:" remains current drive.
ChDir "D:\WINDOWS\SYSTEM"
' On the  Macintosh:
' Changes current folder and current drive.
ChDir "HD:MY FOLDER"
```

See Also **ChDrive Statement**, **MkDir Statement** , **RmDir Statement**, **CurDir Statement**, **Dir Function**

ChDrive Statement

ChDrive Statement

Changes the current drive.

Syntax

ChDrive *drive*

Remarks

The required argument *drive* is a string specifying an existing drive. If the string is empty (""), current drive doesn't change. If *drive* contains more than one symbol, only the first symbol will be used by **ChDrive**.
On the Macintosh **ChDrive** also sets current folder to the root folder of the specified drive.

Example

In the example below the ChDrive statement is used to change the current drive.

```
' In Microsoft Windows:
ChDrive "D" ' Makes drive "D" current.
' On the Macintosh:
' Makes the drive "MY DRIVE" current.
ChDrive "MY DRIVE:"
' Makes drive "MY DRIVE" current. Current folder will be
' the root folder of the drive.
ChDrive "MY DRIVE:MY FOLDER"
```

See Also ChDir Statement, MkDir Statement , RmDir Statement, CurDir Function

Chr Function

Chr Function

Returns a **FixStr** (**String**) value containing the character, associated with the specified character code.

Syntax

Chr[$]([*charcode*])

The optional *charcode* argument is a **Long** that identifies a character. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns an empty string.
The **Chr$** form returns **String** values. The **Chr** form returns **FixStr** values.

Remarks

Numbers from 0 – 31 are the same as standard, nonprintable ASCII codes. For example, **Chr**(10) returns a linefeed character. The normal range for *charcode* is 0 – 255. However, on DBCS systems, the actual range for *charcode* is -32768 to 65535.
If *charcode* is outside the 0-255 range, it will be adjusted to this range using the following formula: *charcode* **Mod** 256.

Example

```
Dim Char
Char = Chr(65)    ' Returns A.
Char = Chr(97)    ' Returns a.
Char = Chr(62)    ' Returns >.
Char = Chr(37)    ' Returns %.
```

See Also Asc Function, Str Function, Type Conversion Functions

Close Statement

Close Statement

Terminates imput/output operations with the file, opened with the **Open** statement.

Syntax

Close [*filenumberlist*]

Optional argument *filenumberlist* can contain one or more file numbers. It's syntax looks the as shown below (*filenumber* is any allowable file number):

[[#]*filenumber*] [, [#]*filenumber*] . . .

Remarks

If the *filenumberlist* argument is omitted, all active files opened with the **Open** statement are closed.

When closing a file opened in the **Output** or **Append** modes, the contents of the last ouptut buffer is added into the file. All buffers, associated with the closed file are cleared.

The **Close** statement breaks relationship between the filename and associated file number.

Example

Here the Close statement is used to close three files that have been opened in the Output mode.

```
Dim I, FileName
For I = 1 To 3                      ' The loop repeats 3 times.
    FileName = "TEST" & I           'Create the filename.
    Open FileName For Output As #I  'Open the file.
    Print #I, "Example."             ' Write a string into the file.
    Next I
Close                               ' Close all 3 open files.
```

See Also **Recording Data in a File**, **End Statement** , **Open Statement**, **Reset Statement**, **Stop Statement**

Comparison Operators

Comparison Operators

Used to compare expressions.

Syntax

result = expression1 comparisonoperator expression2

Comparison operators have these Elements:

Element	**Description**
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.
comparisonoperator	Required; any comparison operator.

Remarks

The following table contains a list of the comparison operators and the conditions that determine whether result is **True** or **False**:

Operator	**True if**	**False if**
< (Less than)	expression1 < expression2	expression1 >= expression2
<= (Less than or equal to)	expression1 <= expression2	expression1 > expression2
> (Greater than)	expression1 > expression2	expression1 <= expression2
>= (Greater than or equal to)	expression1 >= expression2	expression1 < expression2
= (Equal to)	expression1 = expression2	expression1 <> expression2
<> (Not equal to)	expression1 <> expression2	expression1 = expression2

When comparing two expressions, you may not be able to easily determine whether the expressions are being compared as numbers or as strings. The following table shows how the expressions are compared or the result when either expression is not a Variant:

If	**Then**
Both expressions are numeric data types (Byte, Boolean, Integer, Long, Single, Double, or Date)	Perform a numeric comparison.
Both expressions are String	Perform a string comparison.
One expression is a numeric data type and the other is a Variant that is, or can be, a number	Perform a numeric comparison.
One expression is a numeric data type and the other is a string Variant that can't be converted to a number	A Type Mismatch error occurs.

If *expression1* and *expression2* are both **Variant** expressions, their underlying type determines how they are compared. The following table shows how the expressions are compared or the result from the comparison, depending on the underlying type of the **Variant**:

If	Then
Both Variant expressions are numeric	Perform a numeric comparison.
Both Variant expressions are strings	Perform a string comparison.
One Variant expression is numeric and the other is a string	The numeric expression is less than the string expression.

Example

```
Dim MyResult, Var1, Var2
MyResult = (45 < 35)   ' Returns False.
trace MyResult
MyResult = (45 = 45)   ' Returns True.
trace MyResult
MyResult = (4 <> 3)   ' Returns True.
trace MyResult
MyResult = ("5" > "4")   ' Returns True.
trace MyResult
Var1 = "5": Var2 = 4   ' Initialize variables.
MyResult = (Var1 > Var2)   ' Returns True.
trace MyResult
Var1 = 5: Var2 = Empty
MyResult = (Var1 > Var2)   ' Returns True.
trace MyResult
Var1 = 0: Var2 = Empty
MyResult = (Var1 = Var2)   ' Returns True.
trace MyResult
```

See Also [Operators]

& Operator

& Operator

Used to force string concatenation of two expressions.

Syntax

result = *expression1* **&** *expression2*

The **&** operator syntax has these Elements:

Element	Description
result	Required; any **String** or **Variant** variable.

expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If an expression is not a string, it is converted to a **String** variant. The data type of result is **String** if both expressions are string expressions; otherwise, result is a **String** variant. If both expressions are **Null**, result is zero-length string (""). However, if only one expression is **Null**, that expression is treated as a zero-length string ("") when concatenated with the other expression.

Example

This example uses the **&** operator to force string concatenation.

```
Dim MyStr
MyStr = "Hello" & " World" ' Returns "Hello World".
trace MyStr
MyStr = "Check " & 123 & " Check" ' Returns "Check 123 Check".
trace MyStr
```

See Also [Operators]

Language Core Constants

Language Core Constants

Date Format Constants

Constant	**Value**	**Description**
cdbGeneralDate	0	
cdbLongDate	1	
cdbShortDate	2	
cdbLongTime	3	

cdbShortTime	4	

File Attributes Constants

Constant	Value	Description
cdbNormal	0	
cdbReadOnly	1	
cdbHidden	2	
cdbSystem	4	
cdbArchive	32	

Format Function Constants

Constant	Value	Description
cdbUseSystem	0	
cdbSunday	1	
cdbMonday	2	
cdbTuesday	3	
cdbWednesday	4	
cdbThursday	5	
cdbFriday	6	
cdbSaturday	7	

Constant	Value	Description
cdbUseSystem	0	
cdbFirstJan1	1	
cdbFirstFourDays	2	
cdbFirstFullWeek	3	

FormatNumber Function Constants

Constant	**Value**	**Description**
TristateTrue	-1	True
TristateFalse	0	False
TristateUseDefault	-2	Use the setting from the computer's regional settings.

VarType Constants

Constant	**Value**	**Description**
cdbEmpty	0	
cdbNull	1	
cdbInteger	2	
cdbLong	3	
cdbSingle	4	
cdbDouble	5	
cdbDate	7	
cdbString	8	
cdbObject	9	
cdbBoolean	11	
cdbByte	17	

Const Statement

Const Statement

Declares named constants for use in place of literal values.

Syntax

Const *constname* [**As** *type*] = *const_expression*

The **Const** statement syntax has these Elements:

Element	**Description**
constname	Required. Name of the constant; follows standard variable naming conventions.

type	Optional. Data type of the variable; may be Byte, Boolean, Integer, Long, Single, Double, Date, String (for variable-length strings), **String** * *length* (for fixed-length strings), Variant. Use a separate **As** *type* clause for each variable you declare.
const_expression	Required. Constatnt expression; may be numeric constant, string constant, or any combination that includes all arithmetic or logical operators.

Remarks

To combine several constant declarations on the same line, separate each constant assignment with a comma.

You can't use variables, user-defined functions, or intrinsic Basic functions (such as **Chr**) inexpressions assigned to constants.

Note: Constants can make your programs self-documenting and easy to modify. Unlike variables, constants can't be inadvertently changed while your program is running.

If you don't explicitly declare the constant type using **As** *type*, the constant has the data type that is most appropriate for *const_expression*.

Constants declared in a **Sub** or **Function** procedure are local to that procedure. A constant declared outside a procedure is defined throughout the module in which it is declared. You can use constants anywhere you can use an expression.

Example

This example uses the **Const** statement to declare constants for using instead of literal values.

```
' Declare some constants
Const MyVar = 459
Const MyString = "HELP"
' Declare an Integer constant.
Private Const MyInt As Integer = 5
' Declare multiple constants in the  same line.
Const MyStr = "Hello", MyDouble As Double = 3.4567
```

See Also Data Type Summary, Let Statement, Function Statement, Sub Statement

Cos Function

Cos Function

Returns a **Double** specifying the cosine of an angle.

Syntax

Cos([*num*])

The optional *num* argument is a **Double** or any valid numeric expression, specifying the angle in radians. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 1.

Remarks

The **Cos** function takes an angle in radians and returns the ratio of two sides of a right triangle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse.

The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Example

```
Dim MyAngle, MySecant
MyAngle = 1.3   ' Define angle in radians.
MySecant = 1 / Cos(MyAngle)   ' Calculate secant.
```

See Also **Atn Function**, **Sin Function**, **Tan Function**

CurDir Function

CurDir Function

Returns a **FixStr** (**String**) representing the current path.

Syntax

CurDir[$][(*drive*)]

Remarks

The optional *drive* argument is a string expression that specifies an existing drive. If no drive is specified or if *drive* is a zero-length string (""), **CurDir** returns the path for the current drive.

CurDir Function (Apple Power Macintosh)

The optional *drive* argument is a string expression that specifies an existing drive. The **CurDir** function ignores any specified drive and simply returns the path for the current drive.

The **CurDir$** for returns **String** values. The **CurDir** form returns **FixStr** values.

Example

This example uses the **CurDir** function to return the current path.

```
' In Microsoft Windows:
' Assume current path on C drive is  - "C:\WINDOWS\SYSTEM".
' Assume current path on D drive is "D:\EXCEL".
' Assume C is the current drive.
Dim MyPath
MyPath = CurDir      ' Returns "C:\WINDOWS\SYSTEM".
MyPath = CurDir("C") ' Returns "C:\WINDOWS\SYSTEM".
MyPath = CurDir("D") ' Returns "D:\EXCEL".
' For Mac:
' Drive names are ignored. The path for the current disk is returned.
' Assume current path on drive HD is - "HD:MY FOLDER".
' Assume HD is the current drive.
' Assume drive MD also exists on this computer.
Dim MyPath2
MyPath2 = CurDir      ' Returns "HD:MY FOLDER".
MyPath2 = CurDir("HD")' Returns "HD:MY FOLDER".
MyPath2 = CurDir("MD")' Returns "HD:MY FOLDER".
```

See Also	ChDir Statement, ChDrive Statement , MkDir Statement, RmDir Statement

Data Type Summary

Data Type Summary

The following table shows the supported data types, including storage sizes and ranges.

Data type	**Storage size**	**Range**
Byte (byte)	1 byte	From 0 to 255.
Boolean (logical)	2 bytes	True or False.
Integer (integer)	2 bytes	From -32 768 to 32 767

Long (long integer)	4 bytes	From -2 147 483 648 to 2 147 483 647.
Single (single-precision floating point)	4 bytes	From -3,402823E38 to -1,401298E-45 for negative values; from 1,401298E-45 to 3,402823E38 for positive values.

Double
(double-precision floating point)
	8 bytes	From -1,79769313486232E308 to -4,94065645841247E-324 for negative values;
		from 4,94065645841247E-324 to 1,79769313486232E308 for positive values.

Date (date and time)	8 bytes	From 1 January 100 to 31 December 9999
Object (object)	4 bytes	Address that refers to an object
String (variable-length string)	10 bytes + string length	From 0 up to approximately 2 billion (2^31) characters.
String * *n* (FixStr, fixed-length string)	String length	From 1 up to approximately 65 400 characters.
Variant (numeric subtypes)	16 bytes	Any numeric value within the Double range.
Variant (string subtypes)	22 bytes + string length	As for a variable-length string.

See Also

Date Function

Date Function

Returns a **Date** (**String**) containing the current system date.

Syntax

Date[$]()

Remarks

The **Date$** form returns **String** values. The **Date** form returns **Date** values. Use the **Date** statement to set system date.

Example

```
Dim MyDate
MyDate = Date()     ' Assign current system date
```

See Also **Date Statement**, **Format Function**, **Now Function**, **Time Function**, **Time Statement**

Date= Statement

Date= Statement

Sets the current system date.

Syntax

Date = *date*

Remarks

If *date* is a string, **Date** attempts to convert it to a date using the date separators you specified for your system. If it can't be converted to a valid date, an error occurs.

For systems running Microsoft Windows, the required date specification must be a date from January 1, 1980 through December 31, 2079. For systems running Mac OS 9 and later, date must be a date from January 1, 1901 through December 31, 2037.

Note: Changing date is only possible if you have enough rights, required by the system.

Example

```
Dim MyDate
MyDate = #2/17/1995#    ' Assign a date.
Date = MyDate    ' Change system date.
```

See Also **Date Function**, **Time Function**, **Time Statement**

Declare Statement

Declare Statement

Used to declare references to user-defined procedures or external procedures in a dynamic-link library (DLL).

Syntax

Declare Sub *name* [**Lib** "*libname*"] [**Alias** "*aliasname*"] ([*arglist*])

Declare Function *name* [**Lib** "*libname*"] [**Alias** "*aliasname*"] ([*arglist*]) [**As** *type*]

The **Declare** statement syntax has these Elements:

Element	**Description**
Sub	Optional (either **Sub** or **Function** must appear). Indicates that the procedure doesn't return a value.
Function	Optional (either **Sub** or **Function** must appear). Indicates that the procedure returns a value that can be used in an expression.
name	Required. Any valid procedure name. Note that DLL entry points are case sensitive.
Lib	Optional. Indicates that a DLL contains the procedure being declared. The **Lib** clause is required for all external procedures declarations.
libname	Required if **Lib** keyword used. Name of the DLL or code resource that contains the declared procedure.
Alias	Optional. Indicates that the procedure being called has another name in the DLL. This is useful when the external procedure name is the same as a keyword. You can also use **Alias** when a DLL procedure has the same name as a variable, constant, or any other procedure. **Alias** is also useful if any characters in the DLL procedure name aren't allowed by the DLL naming convention.
aliasname	Optional. Name of the procedure in the DLL.
arglist	Optional. List of variables representing arguments that are passed to the procedure when it is called.
type	Optional. Data type of the value returned by a **Function** procedure; may be Byte, Boolean, Integer, Long, Single, Double, Date, String (variable length only), Variant or an object type.

The *arglist* argument has the following syntax and Elements:
[**ByVal** | **ByRef**] *varname* [**As** *type*] [=*defval*]

Element	Description
ByVal	Optional. Indicates that the argument is passed by value. **ByVal** is the default in ConceptDraw Basic.
ByRef	Optional. Indicates that the argument is passed by reference.
varname	Required. Name of the variable representing the argument being passed to the procedure; follows standard variable naming conventions.
type	Optional. Data type of the argument passed to the procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Single**, **Double**, **Date**, **String** (variable length only), **Object**, **Variant** or an object type.
defval	Optional. Constant that determine the value that will be passed to the procedure by default if this argument is omitted.

Remarks

For **Function** procedures, the data type of the procedure determines the data type it returns. You can use an **As** clause following *arglist* to specify the return type of the function. Within *arglist*, you can use an **As** clause to specify the data type of any of the arguments passed to the procedure. In addition to specifying any of the standard data types, you can specify **As Any** in *arglist* to inhibit type checking and allow any data type to be passed to the procedure.

Empty parentheses indicate that the **Sub** or **Function** procedure has no arguments and that ConceptDraw Basic should ensure that none are passed. In the following example, *First* takes no arguments. If you use arguments in a call to *First*, an error occurs:

```
Declare Sub First Lib "MyLib" ()
```

If you include an argument list, the number and type of arguments are checked each time the procedure is called. In the following example, *First* takes one **Long** argument:

```
Declare Sub First Lib "MyLib" (X As Long)
```

Note: You can't have fixed-length strings in the argument list of a **Declare** statement; only variable-length strings can be passed to procedures. Fixed-length strings can appear as procedure arguments, but they are converted to variable-length strings before being passed.

Note: The **cdbNullString** constant is used when calling external procedures, where the external procedure requires a string whose value is zero. This is not the same thing as a zero-length string ("").

Note: If the specified *name* coincides with a keyword, a compilaton error will occur. Make sure you give a unique *name* to the procedure.

Example

This example shows declaring of a user procedure PrintToOutputWindow using the **Declare** instruction before the procedure is called.

```
Declare Sub PrintToOutputWindow(AnyString As String)
' Call a Sub procedure.
Call PrintToOutputWindow("Hello World")
' The above statement causes control to be transferred to the following
' Sub procedure.
Sub PrintToOutputWindow(AnyString As String)
   Trace AnyString    ' Print to the Output window.
End Sub
```

See Also Call Statement , Function Statement , Sub Statement

Dim Statement

Dim Statement

Declares variables and allocates storage space.

Syntax

Dim *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*] [**,** *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*]] . . .

The **Dim** statement syntax has these Elements:

Element	**Description**
varname	Required. Name of the variable; follows standard variable naming conventions.
subscripts	Optional. Dimensions of an array variable; up to 10 multiple dimensions may be declared. The subscripts argument uses the following syntax: *count1*[, *count2*] . . . where *count1*, *count2* are constants, indicating the upper limit of allowable indices for the defined array. The lower limit of allowable indices always equals 0. So, for a one-dimensional array the number of elements can be calculated as *count1*+1 .
New	Optional. Keyword that enables implicit creation of an object. If you use **New** when declaring the object variable, a new instance of the object is created during declaration, so you don't have to use the **Set** statement to assign the object reference. The **New** keyword can't be used to declare

	variables of any intrinsic data type, can't be used to declare instances of dependent objects or objects that don't have built-in constructor.
type	Optional. Data type of the variable; may be Byte, Boolean, Integer, Long, Single, Double, Date, String (for variable-length strings), **String** * *length* (for fixed-length strings), Object, Variant, or an object type. Use a separate **As** *type* clause for each variable you declare.

Remarks

Variables declared with **Dim** at the module level are available to all procedures within the module. At the procedure level, variables are available only within the procedure.

Use the **Dim** statement at module or procedure level to declare the data type of a variable. For example, the following statement declares a variable as an **Integer**.

Dim Number As Integer

Also use a **Dim** statement to declare the object type of a variable. The following declares a variable for a new instance of a database engine.

Dim Eng As New dbEngine

If the **New** keyword is not used when declaring an object variable, the variable that refers to the object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned to an object, the declared object variable has the special value **Nothing**, which indicates that it doesn't refer to any Elementicular instance of an object. When you use the **New** keyword in the declaration, an instance of the object will be created.

You can also use the **Dim** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array.

If you don't specify a data type or object type, the variable is **Variant** by default.

All declared variables except those declared with **New**, take the Empty value, which indicates that they are not initialized.

Tip: It's recommended to place all declarations in the beginning of a module or a procedure. This shortens the time of compilation.

Example

This example shows the **Dim** statement used to declare variables. It also shows the **Dim** statement used to declare arrays.

```
' AnyValue and MyValue are declared as Variant by default.
Dim AnyValue, MyValue
' Explicitly declare a variable of type Integer.
```

```
Dim Number As Integer
' Multiple declarations on a single line. AnotherVar is of type Variant
' because its type is omitted.
Dim AnotherVar, Choice As Boolean, BirthDate As Date
' DayArray is an array of Variants with 51 elements indexed, from
' 0 thru 50
Dim DayArray(50)
' Matrix is a two-dimensional array of integers.
Dim Matrix(3, 4) As Integer
' MyArray is a dynamic array of variants.
Dim MyArray()
```

See Also Data Type Summary, ReDim Statement, Set Statement, Static Statement, Const Statement

Dir Function

Dir Function

Returns a **String** representing the name of a file, directory, or folder that matches a specified pattern or file attribute, or the volume label of a drive.

Syntax

Dir[(*pathname*[, *attributes*])]

The **Dir** function syntax has these Elements:

Element	**Description**
pathname	Optional. String expression that specifies a file name — may include directory or folder, and drive. A zero-length string ("") is returned if *pathname* is not found.
attributes	Optional. Constant or numeric expression, that specifies file attributes. If omitted, returns all files that match *pathname*.

Values

The *attributes* argument settings are:

Constant	**Value**	**Description**
cdbNormal	0	Normal
cdbHidden	2	Hidden

cdbSystem	4	System (Microsoft Windows only)
cdbVolume	8	Volume label; if specified, any other attributes are ignored (Microsoft Windows only)
cdbDirectory	16	Directory or folder
cdbAlias	64	Specified file name is an Alias (Macintosh only)

Note. These constants are specified by the application, that is they can be used anywhere in your code in place of the actual values.

Remarks

When **Dir** is called first time, a path should be specified - otherwise an error will occur. If file attributes are specified, the *pathname* argument is required.
The **Dir** function returns the first file name that matches *pathname*. To get other file names, matching *pathname*, call **Dir** again without arguments. When there are no more matching file names, an empty string ("") is returned. When calling the function after an empty string has been returned, *pathname* must be specified - otherwise an error occurs. You can modify *pathname* at any time. **Dir** can't be called recursively. Calling **Dir** with the **cdbDirectory** attribute doesn't return subfolders subsequently.

Note. As file names are returned in random order, you may store them in an array and then sort.

Example

This example uses the **Dir** function to look for certain files and directories.

```
Dim MyFile, MyPath, MyName
' In Microsoft Windows:
' Returns"WIN.INI" (if exists).
MyFile = Dir("C:\WINDOWS\WIN.INI")
' Returns a file name with specified extension. If more than one *.INI file
exist
' returns the first file found.
MyFile = Dir("C:\WINDOWS\*.INI")
' Call Dir again with no arguments to get the next *.INI file
' located in the same directory.
MyFile = Dir
' Returns the first found *.TXT file with hidden attribute.
MyFile = Dir("*.TXT", cdbHidden)
' Returns the list of directories on drive C:.
MyPath = "c:\"                     ' Specify path.
MyName = Dir(MyPath, cdbDirectory) ' Retrieve the  first entry.
Do While MyName <> ""       ' Start the  loop.
    ' Ignore the current directory and the encompassing directory.
    If MyName <> "." And MyName <> ".." Then
    ' Use bitwise comparison to make sure MyName is a directory.
        If (GetAttr(MyPath & MyName) And cdbDirectory) = cdbDirectory Then
            Trace MyName    ' Display entry only if it represents a directory
              End If
    End If
    MyName = Dir            ' Get next entry.
Loop
```

See Also [Инструкция ChDir](), [Функция CurDir]()

/ Operator

/ Operator

Used to divide two numbers and return a floating-point result.

Syntax

result = number1 / number2

The + operator syntax has these Elements:

Element	**Description**
result	Required; any numeric variable.
number1	Required; any numeric expression.
number2	Required; any numeric expression.

Remarks

The data type of result is usually a Double or a Double variant. The following are exceptions to this rule:

If	**Then *result* is**
Both expressions are **Byte** variants,	a **Byte** variant unless it overflows its legal range; in which case, result is a Variant containing a **Integer**.
Both expressions are **Integer** variants,	a **Integer** variant unless it overflows its legal range; in which case, result is a Variant containing a **Long**.
Both expressions are **Long**, **Single** variants,	a **Long**, **Single** variant unless it overflows its legal range; in which case, result is a Variant containing a **Double**.

If one expressions are **Null** or **Empty** expressions, result is 0.

Example

This example uses the / operator to perform floating-point division.

```
Dim MyValue
MyValue = 10 / 4    ' Returns 2.5.
trace MyValue
MyValue = 10 / 3    ' Returns 3.333333.
trace MyValue
```

See Also Operators

Do...Loop Statement

Do...Loop Statement

Repeats a block of statements while a condition is **True** or until a condition becomes **True**.

Syntax

Do [{**While** | **Until**} *condition*]
[*statements*]
[**Exit Do**]
[*statements*]

Loop

Or, you can use this syntax:
Do
[*statements*]
[**Exit Do**]
[*statements*]

Loop [{**While** | **Until**} *condition*]

The **Do Loop** statement syntax has these Elements:

Element	**Description**
condition	Optional. Expression that is **True** or **False**.
statements	One or more statements that are repeated while, or until, *condition* is **True**.

Remarks

Any number of **Exit Do** statements may be placed anywhere in the **Do…Loop** as an alternate way to exit a **Do…Loop**. **Exit Do** is often used after evaluating some condition, for example, **If…Then**, in which case the **Exit Do** statement transfers control to the statement immediately following the **Loop**.

When used within nested **Do…Loop** statements, **Exit Do** transfers control to the loop that is one nested level above the loop where **Exit Do** occurs.

Example

This example shows how **Do...Loop** statements can be used. The inner **Do...Loop** statement loops 10 times, sets the value of the flag to **False**, and exits prematurely using the **Exit Do** statement. The outer loop exits immediately upon checking the value of the flag.

```
Dim Check, Counter
Check = True: Counter = 0   ' Initialize variables.
Do   ' Outer loop.
   Do While Counter < 20   ' Inner loop.
      Counter = Counter + 1   ' Increment Counter.
      If Counter = 10 Then   ' If condition is True.
         Check = False   ' Set value of flag to False.
         Exit Do   ' Exit inner loop.
      End If
   Loop
Loop Until Check = False   ' Exit outer loop immediately.
```

See Also Exit Statement , For...Next Statement , While...Wend Statement

End Statement

End Statement

Ends a procedure or block.

Syntax

End

End Function
End If

End Select

End Sub

The **End** statement syntax has these forms:

Statement	**Description**
End	Terminates running the script. It's not required, but can be placed anywhere in the program for closing files, opened with **Open**, and for clearing variables.
End Function	Required statement to close the **Function** construction.
End If	Required statement to close the **IfEThenEElse** construction.
End Select	Required statement to close the **Select Case** construction.
End Sub	Required statement to close the **Sub** construction.

Remarks

The **End** statement resets all variables at the module level and all static local variables in all modules. Current-level script stops running, which causes script on lower execution levels stop running too. For instance, if the **End** statement was performed in the document's script, scripts at the Page and Shape level immediately stop running.

If you need to save values of global variables and leave the program waiting for its procedure calls, you should use the **Stop** statement.

Note: The **End** statement immediately stops execution of the script. Files open with the **Open** statement are closed, and memory used by the program is cleared.

Example

In the example below the End statement is used to terminate the program if the user provides an incorrect password.

```
Sub EndSample()
    Dim Pword
    const PassWord = "password"
    Pword = InputBox("Enter password")
    If Pword <> PassWord Then
         MsgBox "Illegal password"
         End  ' Stops program execution
    End If
End Sub
```

See Also	Function Statement, If ... Then ... Else Statement, Select Case Statement , Stop_Statement, Sub Statement

Enum Statement

Enum Statement

Declares enumeration.

Syntax

Enum *Name*
  constName1 [= *value1*]
  [*constName2* [= *value2*]]
  ...
End Enum

The **Enum** statement syntax has these Elements:

Element	Description
Enum	Required; keyword.
constName1, *constName2*...	First is required. Names of the enumeration constants.
value1, *value2*...	Optional. The enumeration constants values.

Remarks

Declaring enumerations is a quick way to declare several named constants. If their values are not assigned directly, they start from 0 and increase by 1 every next *constName*. If the value of a certain constant is set, next value will differ by 1.

Example

```
Enum numbers
 zero   ' = 0
 five = 5 ' = 5
 six   ' = 6
End Enum
```

See Also Const Statement

EOF Function

EOF Function

Returns the **Boolean** value **True** when the end of a file has been reached.

Syntax

EOF(*filenumber*)

The required *filenumber* argument is an **Integer** containing any valid file number.

Remarks

Use **EOF** to avoid the error generated by attempting to get input past the end of a file.

The **EOF** function returns **False** until the end of the file has been reached. With files opened for **Random** or **Binary** access, **EOF** returns **False** until the last executed **Get** statement is unable to read an entire record.

With files opened for **Binary** access, an attempt to read through the file using the **Input** function until **EOF** returns **True** generates an error. Use the **LOF** and **Loc** functions instead of **EOF** when reading binary files with **Input**, or use **Get** when using the **EOF** function.

Example

This example uses the **EOF** function to detect the end of a file.
This example assumes that TESTFILE is a text file with a few lines of text.

```
Dim InputData
Open "TESTFILE" For Input As #1 ' Open file for input.
Do While Not EOF(1)             ' Check for end of file.
    Line Input #1, InputData    ' Read line of data.
    Trace InputData             ' Print to the Output window.
Loop
Close #1                        ' Close file.
```

See Also **Get Statement** , **Open Statement** , **Loc Function**, **LOF Function**

EQV Operator

EQV Operator

Used to perform a logical equivalence on two expressions.

Syntax

result = *expression1* **Eqv** *expression2*

The Eqv operator syntax has these Elements:

Element	Description
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

The following table illustrates how result is determined:

If expression1 is	And expression2 is	The result is
True	True	True
True	False	False
False	True	False
False	False	True

The Eqv operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following table:

If expression1 is	And expression2 is	The result is
0	0	1
0	1	0
1	0	0
1	1	1

Example

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null    ' Initialize variables.
MyCheck = A > B Eqv B > C    ' Returns True.
trace MyCheck
MyCheck = B > A Eqv B > C    ' Returns False.
trace MyCheck
MyCheck = A > B Eqv B > D    ' Returns True.
trace MyCheck
MyCheck = A Eqv B    ' Returns -3 (bitwise comparison).
trace MyCheck
```

See Also Operators

Erase Statement

Erase Statement

Reinitializes the elements of arrays

Syntax

Erase *arraylist*

The required *arraylist* argument is one or more comma-delimited array variables to be erased.

Remarks

Erase sets the elements of an array to **Empty**.

Example

This example uses the **Erase** statement to reinitialize the elements of arrays.

```
' Declare array variables.
Dim NumArray(10) As Integer         ' Integer array.
Dim StrVarArray(10) As String       ' Variable-string array.
Dim StrFixArray(10) As String * 10  ' Fixed-string array.
Dim VarArray(10) As Variant         ' Variant array.
Erase NumArray      ' Each element is set to Empty, which is equivalent to 0.
Erase StrVarArray   ' Each element is set to Empty, which is equivalent to
NULL.
Erase StrFixArray   ' Each element is set to Empty, which is equivalent to an
empty string"".
Erase VarArray      ' Each element is set to Empty.
```

See Also Dim Statement, ReDim Statement, Static Statement

Erl Function

Erl Function

Returns the script code line number where the most recent run-time error occured.

Syntax

Erl()

Remarks

Use the **Erl** function to determine the line number of the source code where a run-time error occured. Usually it may be needed for debugging in an error handler defined by the **On Error** statement.

Example

```
Sub ErlFuncDemo()
   On Error GoTo ErrorHandler   ' Enable error-handling routine.
   Open "TESTFILE" For Output As #1   ' Open file for output.
   Kill "TESTFILE"   ' Attempt to delete open file.
   Dim d As Double
   d = 10 / sin(0)   ' "Division by zero" error
   d = 20 / cos(0)
   Trace d
Exit Sub      ' Exit to avoid handler.
ErrorHandler:   ' Error-handling routine.
   errNumber = Err() ' Get error number
   errLine = Erl() ' Get source code line
   Trace "ErrorNumber " & errNumber & " at line " & errLine
   Select Case errNumber   ' Evaluate error number.
      Case 55, 75   ' "File already open" or "Path/File access error" error.
         Trace """File already open"" or ""Path/File access error"" error"
         Close #1   ' Close open file.
      Case Else
         ' Handle other situations here...
         Resume Next
   End Select
   Resume   ' Resume execution at same line that caused the error.
End Sub
```

See Also Err Function , Error$ Function , On Error Statement , Trappable Errors

Error$ Function

Error$ Function

Returns the error message that corresponds to a given error number.

Syntax

Error$([*errornumber*])

The optional *errornumber* argument can be any valid error number. If *errornumber* is a valid error number, but is not defined or *errornumber* is not valid, **Error** returns the string "Undefined internal error". If *errornumber* is omitted, the message corresponding to the most recent run-time error is returned.

Example

This example uses the **Error** function to print error messages that correspond to the specified error numbers.

```
Dim ErrorNumber
For ErrorNumber = 1 To 99   ' Loop through values 1 - 99.
   Trace Error(ErrorNumber)    ' Print error to Output window.
Next ErrorNumber
```

See Also Erl Function , Err Function , Trappable Errors

Error Statement

Error Statement

Simulates the occurrence of an error.

Syntax

Error *errornumber*

The required *errornumber* can be any valid error number.

Remarks

The **Error** statement is used to generate run-time errors.

Example

This example uses the **Error** statement to simulate error number 11.

```
On Error Resume Next   ' Defer error handling.
Error 11   ' Simulate the "Division by zero" error.
```

See Also Erl Function , Err Function , Error$ Function , On Error Statement , Resume Statement , Trappable Errors

Err Function

Err Function

Returns the error number corresponding to the most recent run-time error.

Syntax

Err()

Remarks

The **Err** function is normally used to determine the number of an occured run-time error. Usually it may be needed in an error handler defined by the **On Error** statement.

Example

This example shows how the **Err** function is used in the ErrorHandler error-handling routine:

```
Sub ErrFuncDemo()
   On Error GoTo ErrorHandler   ' Enable error-handling routine.
   Open "TESTFILE" For Output As #1   ' Open file for output.
   Kill "TESTFILE"   ' Attempt to delete open file.
   Dim d As Double
   d = 10 / sin(0)   ' "Division by zero" error
   d = 20 / cos(0)
   Trace d
Exit Sub      ' Exit to avoid handler.
ErrorHandler:   ' Error-handling routine.
   errNumber = Err() ' Get error number
   Trace "ErrorNumber " & errNumber
   Select Case errNumber   ' Evaluate error number.
      Case 55, 75   ' "File already open" or "Path/File access error" error.
         Trace """File already open"" or ""Path/File access error"" error"
         Close #1   ' Close open file.
      Case Else
         ' Handle other situations here...
         Resume Next
   End Select
   Resume   ' Resume execution at same line that caused the error.
End Sub
```

See Also Erl Function , Error$ Function , On Error Statement , Trappable Errors

Exit Statement

Exit Statement

Exits a block of **Do…Loop**, **For...Next**, **Function**, or **Sub** code.

Syntax

Exit Do

Exit For

Exit Function
Exit Sub

The **Exit** statement syntax has these forms:

Statement	**Description**
Exit Do	Provides a way to exit a **Do...Loop** statement. It can be used only inside a **Do...Loop** statement. **Exit Do** transfers control to the statement following the **Loop** statement. When used within nested **Do...Loop** statements, **Exit Do** transfers control to the loop that is one nested level above the loop where **Exit Do** occurs.
Exit For	Provides a way to exit a **For** loop. It can be used only in a **For...Next** loop. **Exit For** transfers control to the statement following the **Next** statement. When used within nested **For** loops, **Exit For** transfers control to the loop that is one nested level above the loop where **Exit For** occurs.
Exit Function	Immediately exits the **Function** procedure in which it appears. Execution continues with the statement following the statement that called the **Function**.
Exit Sub	Immediately exits the **Sub** procedure in which it appears. Execution continues with the statement following the statement that called the **Sub** procedure.

Remarks

Do not confuse **Exit** statements with **End** statements. **Exit** does not define the end of a code block.

Example

This example uses the **Exit** statement to exit a **For...Next** loop, a **Do...Loop**, and a **Sub** procedure.

```
Sub ExitStatementDemo()
Dim I As Integer, MyNum As Integer
   Do            ' Set up infinite loop.
      For I = 1 To 1000   ' Loop 1000 times.
         MyNum = CInt(Rnd() * 1000)    ' Generate random numbers.
         Select Case MyNum   ' Evaluate random number.
            Case 7:
   Trace "Exit For"
   Exit For   ' If 7, exit For...Next.
            Case 29:
   Trace "Exit Do"
   Exit Do   ' If 29, exit Do...Loop.
            Case 54:
   Trace "Exit Sub"
   Exit Sub   ' If 54, exit Sub procedure.
         End Select
      Next I
   Loop
End Sub
ExitStatementDemo()
```

See Also Do...Loop Statement , End Statement , For...Next Statement , Function Statement , Stop Statement , Sub Statement

Exp Function

Exp Function

Returns a **Double** specifying **e** (the base of natural logarithms) raised to a power.

Syntax

Exp([*num*])

The optional *num* argument is a **Double** or any valid numeric expression. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 1.

Remarks

If the value of number exceeds 709.782712893, an error occurs. The constant **e** is approximately 2.718282.

Note The **Exp** function complements the action of the **Log** function and is sometimes referred to as the antilogarithm.

Example

```
Dim MyAngle, MyHSin
' Define angle in radians.
MyAngle = 1.3
' Calculate hyperbolic sine.
MyHSin = (Exp(MyAngle) - Exp(-1 * MyAngle)) / 2
```

See Also **Log Function**

^ , ** Operators

^ , ** Operators

Used to raise a number to the power of an exponent.

Syntax

result = expression1 ^expression2

*result = expression1 **expression2*

The ^ operator (** operator) syntax has these Elements:

Element	**Description**
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

A number can be negative only if exponent is an integer value. When more than one exponentiation is performed in a single expression, the ^ operator is evaluated as it is encountered from right to left.
Usually, the data type of result is a **Double** or a **Variant** containing a **Double**. However, if either number or exponent is a **Null** expression, result is 1.

Example

This example uses the ^ operator to raise a number to the power of an exponent.

```
Dim MyValue
MyValue = 2 ^ 2    ' Returns 4.
trace MyValue
MyValue = 3 ^ 3 ^ 2    ' Returns 19683.
trace MyValue
MyValue = (-5) ^ 3    ' Returns -125.
trace MyValue
```

See Also [Operators]

FileAttr Function

FileAttr Function

Returns a **Long** representing the file mode for files opened using the **Open** statement.

Syntax
FileAttr(*filenumber*, *returntype*)

The **FileAttr** function syntax has these named arguments:

Element	**Description**
filenumber	Required; *Integer*. Any validfile number.
returntype	Required; *Integer*. Number indicating the type of information to return. Specify 1 to return a value indicating the file mode.

Return Values

When the *returntype* argument is 1, the following return values indicate the file access mode:

Mode	**Value**
Input	1
Output	2
Random	4
Append	8
Binary	32

Example

This example uses the **FileAttr** function to return the file mode of an open file.

```
Dim FileNum, Mode
FileNum = FreeFile()                       ' Assign file number.
Open "TESTFILE" For Append As FileNum ' Open file.
Mode = FileAttr(FileNum, 1)                ' Returns  8 (Append file mode).
Close FileNum                              ' Close file.
Trace Mode
```

See Also Open Statement, SetAttr Statement , GetAttr Function

FileCopy Statement

FileCopy Statement

Copies the file.

Syntax
FileCopy *source*, *destination*

The **FileCopy** statement syntax contains the following Elements:

Element	Description
source	Required. A string indicating the name of the file to be copied. Path may include folder and disk name.
destination	Required. A string that specifies the name of the resulting file. Path may include folder and disk name.

Remarks

An attempt to copy an open file with **FileCopy** will generate an error.

Example

In this example FileCopy is used to create a copy of a file. Assume that the file SRCFILE exists and is not empty.

```
Dim SourceFile, DestinationFile
SourceFile = "SRCFILE"                     ' Source File.
DestinationFile = "DESTFILE"               ' Destination File.
```

```
FileCopy SourceFile, DestinationFile  ' File Copy.
```

See Also Kill Statement, Name Statement

FileDateTime Function

FileDateTime Function

Returns a **Variant** (**Date**) value that indicates the date and time when a file was created or last modified.

Syntax
FileDateTime(*path*)

Remarks

The required *path* argument is a string expression that specifies a file name. The *path* may include the directory or folder, and the drive.

Example

This example uses the **FileDateTime** function to determine the date and time a file was created or last modified. The format of the date and time displayed is based on the locale settings of your system.

```
Dim MyStamp
' Assume TESTFILE was created on February 12, 1993 at 16:35:47.
' Assume Russian locale settings.
MyStamp = FileDateTime("TESTFILE") ' Returns"12.02.93 16:35:47".
Trace MyStamp
```

See Also FileLen Function, GetAttr Function , VarType Function

FileLen Function

FileLen Function

Returns a **Long** specifying the length of a file in bytes.

Syntax

FileLen(*path*)

The required *path* argument is a string expression that specifies a file. The *path* may include the directory or folder, and the drive.

Remarks

If the specified file is open when the **FileLen** function is called, the value returned represents the size of the file immediately before it was opened.

Note. To obtain the length of an open file, use the **LOF** function.

Example

This example uses the **FileLen** function to return the length of a file in bytes.
For purposes of this example, assume that TESTFILE is a file containing some data.

```
Dim MySize
MySize = FileLen("TESTFILE")  ' Returns file length (bytes).
Trace MySize
```

See Also **FileDateTime Function**, **GetAttr Function** , **LOF Function**

FormatDateTime Function

FormatDateTime Function

Returns a **FixStr** value formatted as a date or time.

Syntax

FormatDateTime([*Date*[,*NamedFormat*]])

The **FormatDateTime** function syntax has these Elements:

Element	Description
Date	Optional. Date expression to be formatted.
NamedFormat	Optional. Numeric value that indicates the date/time format used. If omitted, **cdbGeneralDate** is used.

If the *Date* argument is omitted, is a non-initialized variable, or **Null**, the function returns zero time and/or date.

Settings

The *NamedFormat* argument has the following settings:

Name	Setting	Description
cdbGeneralDate	0	Display a date and/or time. If there is a date Element, display it as a short date. If there is a time Element, display it as a long time. If present, both Elements are displayed.
cdbLongDate	1	Display a date using the long date format specified in your computer's regional settings.
cdbShortDate	2	Display a date using the short date format specified in your computer's regional settings.
cdbLongTime	3	Display a time using the time format specified in your computer's regional settings.
cdbShortTime	4	Display a time using the 24-hour format (hh:mm).

See Also **FormatNumber Function**, **Format Function**

FormatNumber Function

FormatNumber Function

Returns a **FixStr** value formatted as a number.

Syntax

FormatNumber([*Expression*[,*NumDigitsAfterDecimal* [,*IncludeLeadingDigit* [,*UseParensForNegativeNumbers* [,*GroupDigits*]]]]])

If the *Expression* argument is omitted, is a non-initialized variable, or **Null**, the function returns 0.

The **FormatNumber** function syntax has these Elements:

Element	Description
Expression	Optional. Expression to be formatted.
NumDigitsAfterDecimal	Optional. Numeric value indicating how many places to the right of the decimal are displayed. Default value is -1, which indicates that the computer's regional settings are used.
IncludeLeadingDigit	Optional. Tristate constant that indicates whether or not a leading zero is displayed for fractional values. See Settings section for values.
UseParensForNegativeNumbers	Optional. Tristate constant that indicates whether or not to place negative values within parentheses. See Settings section for values.
GroupDigits	Optional. Tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the computer's regional settings. See Settings section for values.

Settings

The *IncludeLeadingDigit*, *UseParensForNegativeNumbers*, and *GroupDigits* arguments have the following settings:

Constant	Value	Description
TristateTrue	-1	True
TristateFalse	0	False
TristateUseDefault	-2	Use the setting from the computer's regional settings.

Remarks

When one or more optional arguments are omitted, the values for omitted arguments are provided by the computer's regional settings. If the *expression* argument is omitted, it's value is considered 0.

Note All settings information comes from the **Regional Settings Number** tab. In *expression* point (**.**) is used as decimal separator.

Example

The following example uses the FormatNumber function to format a number to have four decimal places:

```
Dim MyAngle, MySecant, MyNumber
MyAngle = 1.3   ' Define angle in radians.
MySecant = 1 / Cos(MyAngle)   ' Calculate secant.
MyNumber = FormatNumber(MySecant,4)    ' Format MySecant to four decimal
places.
```

See Also **Format Function**, **FormatDateTime Function**

Format Function

Format Function

Returns a **FixStr** (**String**) value containing an expression formatted according to instructions contained in a format expression.

Syntax

Format[$]([*expression*[, *format*[, *firstdayofweek*[, *firstweekofyear*]]]])

The **Format** function syntax has these Elements:

Element	**Description**
expression	Optional. Any valid expression.
format	Optional. A valid named or user-defined format expression.
firstdayofweek	Optional. A constant that specifies the first day of the week.
firstweekofyear	Optional. A constant that specifies the first week of the year.

If the *expression* argument is omitted, is a non-initialized variable, or **Null**, the function returns an empty string.

Settings

The *firstdayofweek* argument has these settings:

Constant	Value	Description
cdbUseSystem	0	Use NLS API setting.
cdbSunday	1	Sunday (default)
cdbMonday	2	Monday.
cdbTuesday	3	Tuesday.
cdbWednesday	4	Wednesday.
cdbThursday	5	Thursday.
cdbFriday	6	Friday.
cdbSaturday	7	Saturday.

The *firstweekofyear* argument has these settings:

Constant	Value	Description
cdbUseSystem	0	Use NLS API setting.
cdbFirstJan1	1	Start with week in which January 1 occurs (default).
cdbFirstFourDays	2	Start with the first week that has at least four days in the year.
cdbFirstFullWeek	3	Start with the first full week of the year.

Remarks

To Format	Do This
Numbers	Use predefined named numeric formats or create user-defined numeric formats.

Strings	Create your own user-defined string formats.
Dates and times	Use predefined named date/time formats or create user-defined date/time formats.

If you try to format a number without specifying *format*, **Format** provides functionality similar to the **Str** function. However, positive numbers formatted as strings using **Format** don’t include a leading space reserved for the sign of the value; those converted using **Str** retain the leading space.

Named Date/Time Formats

The following table identifies the predefined date and time format names:

Format Name	**Description**
Long Date	Display a date according to your system's long date format.
Medium Date	Display a date using the medium date format appropriate for the language version of the host application.
Short Date	Display a date using your system's short date format.
Long Time	Display a time using your system's long time format; includes hours, minutes, seconds.
Medium Time	Display time in 12-hour format using hours and minutes and the AM/PM designator.
Short Time	Display a time using the 24-hour format, for example, 17:45.

User-Defined Date/Time Formats

The following table identifies characters you can use to create user-defined date/time formats:

Character	**Description**
(**:**)	Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by your system settings.
(**/**)	Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by your system settings.

c	Display the date as ddddd and display the time as ttttt, in that order. Display only date information if there is no fractional Element to the date serial number; display only time information if there is no integer portion.
d	Display the day as a number without a leading zero (1-31).
dd	Display the day as a number with a leading zero (01-31).
ddd	Display the day as an abbreviation (Sun-Sat).
dddd	Display the day as a full name (Sunday-Saturday).
ddddd	Display the date as a complete date (including day, month, and year), formatted according to your system's short date format setting. The default short date format is `m/d/yy`.
dddddd	Display a date serial number as a complete date (including day, month, and year) formatted according to the long date setting recognized by your system. The default long date format is `mmmm dd, yyyy`.
w	Display the day of the week as a number (1 for Sunday through 7 for Saturday).
ww	Display the week of the year as a number (1-54).
m	Display the month as a number without a leading zero (1-12). If `m` immediately follows `h` or `hh`, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01-12). If `m` immediately follows `h` or `hh`, the minute rather than the month is displayed.
mmm	Display the month as an abbreviation (Jan-Dec).
mmmm	Display the month as a full month name (January-December).
q	Display the quarter of the year as a number (1-4).
y	Display the day of the year as a number (1-366).
yy	Display the year as a 2-digit number (00-99).
yyyy	Display the year as a 4-digit number (100-9999).
h	Display the hour as a number without leading zeros (0-23).
Hh	Display the hour as a number with leading zeros (00-23).
N	Display the minute as a number without leading zeros (0-59).

Nn	Display the minute as a number with leading zeros (00-59).
S	Display the second as a number without leading zeros (0-59).
Ss	Display the second as a number with leading zeros (00-59).
t t t t t	Display a time as a complete time (including hour, minute, and second), formatted using the time separator defined by the time format recognized by your system. A leading zero is displayed if the leading zero option is selected and the time is before 10:00 A.M. or P.M. The default time format is `h:mm:ss`.
AM/PM	Use the 12-hour clock and display an uppercase AM with any hour before noon; display an uppercase PM with any hour between noon and 11:59 P.M.
A/P	Use the 12-hour clock and display an uppercase A with any hour before noon; display an uppercase P with any hour between noon and 11:59 P.M.
AMPM	Use the 12-hour clock and display the AM string literal as defined by your system with any hour before noon; display the PM string literal as defined by your system with any hour between noon and 11:59 P.M. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as defined by your system settings. The default format is AM/PM.

User-Defined String Formats

You can use the following characters for formatting strings. In this case the string must begin with one these characters. If you use more than one characters, (<) or (>) must go first, then (**!**) and then other characters. If the leading character in *format* is a character not listed in the table below, the result will be indefinite.

Character	**Description**
@	Placeholder for a character from *expression*. Display a character from expression or a space. If *expression* has a character in the position where the at symbol (**@**) appears in the format string, display it; otherwise, display a space in that position. Below is an example of how words are aligned to the right. The underline (**_**) sign is used to imitate a space. It looks as if the resulting line were filled from right to left with the help of the *format* argument: *Format("Wasya", "@@@@@@'s") Format("Dime", "@@@@@@'s") Format("Ruslan", "@@@@@@'s") Return: "_Wasya's" Return: "__Dime's" Return: "Ruslan's"*

	If the number of at signs (**@**) is less than the number of characters in *expression*, all symbols defined in *expression* will be displayed. The the example below the result will be the whole *expression*, and non-special characters are placed in the same positions, as in *format*.
	`Format("Fine Scotch Whisky", "@@@@ Old")`
	`Return: "Fine Old Scotch Whisky"`

&	Placeholder for a character from *expression*. Display a character from *expression* or nothing. If the string has a character in the position where the ampersand (**&**) appears, display it; otherwise, display nothing. Below is an example that aligns sentences to the left:
	`Format("Red", "&&&&& Hat")`
	`Format("Green", "&&&&& Hat")`
	`Format("Blue", "&&&&& Hat")`
	`Return: "Red Hat"`
	`Return: "Green Hat"`
	`Return: "Blue Hat"`
	`Format("7305305", "Your phonenumber is &&&&&&&&&&&&&. Isn't it ?")`
	`Format("8(0482)266576", "Your phonenumber is &&&&&&&&&&&&&. Isn't it ?")`
	`Return: "Your phonenumber is 7305305. Isn't it ?"`
	`Return: "Your phonenumber is 8(0482)266575. Isn't it ?"`
	If the number of ampersand (**&**) characters is less than the number of characters in *expression*, the result is equivalent to (**@**) in the same case.
	`Format("RockRoll", "&&&&\&")`
	`Return: "Rock&Roll"`

<	Force lowercase. Display all characters in lowercase format.
>	Force uppercase. Display all characters in uppercase format.

!	Force left to right fill of placeholders. The default is to fill placeholders from right to left. When the number of at signs (**@**) in *format* is greater than the number of characters in *expression*, the resulting expression is filled from left to right instead of right to left. That is, spaces are added to the right. Example:
	
	`Format("September","!@@@@@@@@@@@9")`
	`Format("October",  "!@@@@@@@@@@10")`
	`Format("November", "!@@@@@@@@@@11")`
	`Return: "September__9"`
	`Return: "October___10"`
	`Return: "November__11"`
	If the number of at and ampersand signs (**@**) (**&**) is less than the number of signs in *expression*, placeholders are filled right to left, and "extra" characters are not displayed. Example:
	`Format("www.conceptdraw.com","!@@@")`
	`Format("www.abc.net","!@@@")`
	`Return: "com"`
	`Return: "net"`

User-Defined Numeric Formats

Format Name	**Description**
None	Display the number with no formatting.
(**0**)	Digit placeholder. Display a digit or a zero. If theexpression has a digit in the position where the **0** appears in the format string, display it; otherwise, display a zero in that position. If the number has fewer digits than there are zeros (on either side of the decimal) in the format expression, display leading or trailing zeros. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, round the number to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, display the extra digits without modification.
(**#**)	Digit placeholder. Display a digit or nothing. If the expression has a digit in the position where the **#** appears in the format string, display it; otherwise, display nothing in that position. This symbol works like the 0 digit placeholder, except that leading and trailing zeros aren't displayed if the number has the same or fewer digits than there are **#** characters on either side of the decimal separator in the format expression.
(**.**)	Decimal placeholder. In somelocales, a comma is used as the decimal separator. The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only number signs to the left of this symbol, numbers smaller than 1 begin with a decimal separator. To display a leading zero displayed with fractional numbers, use 0 as the first digit placeholder to the left of the decimal separator. The actual character used as a decimal placeholder in the formatted output depends on the Number Format recognized by your system.
(\)	Display the next character in the format string. To display a character that has special meaning as a literal character, precede it with a backslash (\). The backslash itself isn't displayed. Using a backslash is the same as enclosing the next character in double quotation marks. To display a backslash, use two backslashes (\\).

Named Numeric Formats

Format Name	**Description**
Scientific	Use standard scientific notation.

Remarks

If you want to form a string but don't want some characters in *format* to be treated as special characters, put "\" before each such character.

"\" is not required for formatting numbers.

Example

```
Dim MyTime, MyDate, MyStr
MyTime = #17:04:23#
MyDate = #January 27, 1993#
' Returns current system time in the system-defined long time format.
MyStr = Format(Time, "Long Time")
' Returns current system date in the system-defined long date format.
MyStr = Format(Date, "Long Date")
MyStr = Format(MyTime, "h:m:s")              ' Returns "17:4:23".
MyStr = Format(MyTime, "hh:mm:ss AM/PM")     ' Returns "05:04:23 PM".
MyStr = Format(MyDate, "dddd, mmm d yyyy")   ' Returns "Wednesday,  Jan 27
1993".
' If format is not supplied, a string is returned.
MyStr = Format(23)    ' Returns "23".
' User-defined formats.
MyStr = Format(15, "She is # years old") ' Returns "She is 15 years old".
MyStr = Format(346.3, "+0.000")          ' Returns "+346.300".
MyStr = Format("HELLO", "<")             ' Returns "hello".
MyStr = Format("NewYork", ">!&&&&")   ' Returns "YORK"
MyStr = Format("Black Sea", "&&&&& Nice")' Returns "Black Nice Sea"
MyStr = Format("32 June", "This is very strange date: &&&&&&&&&&, isn't it?")
                          'Returns  "This is very strange date: 32 June, isn't
it?"
```

See Also   Type Conversion Functions

For...Next Statement

For...Next Statement

Repeats a group of statements a specified number of times.

Syntax

For *counter* = *start* **To** *end* [**Step** *step*]
[*statements*]
[**Exit For**]
[*statements*]

Next [*counter*]

The **ForENext** statement syntax has these Elements:

Element	Description
counter	Required. Numeric variable used as a loop counter. The variable can't be a Boolean or an array element.
start	Required. Initial value of *counter*.
end	Required. Final value of *counter*.
step	Optional. Amount *counter* is changed each time through the loop. If not specified, *step* defaults to one.
statements	Optional. One or more statements between **For** and **Next** that are executed the specified number of times.

Remarks

The *step* argument can be either positive or negative. The value of the step argument determines loop processing as follows:

Value	Loop executes if
Positive or 0	*counter* <= *end*
Negative	*counter* >= *end*

After all statements in the loop have executed, *step* is added to *counter*. At this point, either the statements in the loop execute again (based on the same test that caused the loop to execute initially), or the loop is exited and execution continues with the statement following the **Next** statement.

Note: Changing the value of *counter* while inside a loop can make it more difficult to read and debug your code.

Any number of **Exit For** statements may be placed anywhere in the loop as an alternate way to exit. **Exit For** is often used after evaluating of some condition, for example **If...Then**, and transfers control to the statement immediately following **Next**.

You can nest **For...Next** loops by placing one **For...Next** loop within another. Give each loop a unique variable name as its counter. The following construction is correct:

```
For I = 1 To 10
   For J = 1 To 10
      For K = 1 To 10
      ...
      Next K
   Next J
Next I
```

Note: If you omit *counter* in a **Next** statement, execution continues as if *counter* were included. In fact, the "**Next** *counter*" construction in ConceptDraw Basic is supported only for the purpose of compatibility with other popular releases of BASIC language. *Counter* is not necessary in the **Next** statement, and ConceptDraw Basic doesn't use it. ConceptDraw Basic keeps track of nested **For...Next** loops using just the keywords **For** and **Next**. Each **Next** matches the most recent **For**.

Example

This example uses the **For...Next** statement to create a string that contains 10 instances of the numbers 0 through 9, each string separated from the other by a single space. The outer loop uses a loop counter variable that is decremented each time through the loop.

```
Dim Words, Chars, MyString
For Words = 10 To 1 Step -1     ' Set 10 repetitions.
   For Chars = 0 To 9    ' Set 10 repetitions.
      MyString = MyString & Chars    ' Append number to string.
   Next    ' Increment Chars counter
   MyString = MyString & " "    ' Append a space.
Next
Trace ">" & MyString & "<"
```

See Also Do..Loop Statement, Exit Statement , While...Wend Statement

FreeFile Function

FreeFile Function

Returns an **Integer** representing the next file number available for use by the **Open** statement.

Syntax

FreeFile[(*rangenumber*)]

Remarks

The optional *rangenumber* argument is a **Variant** that specifies the range from which the next free file number is to be returned. Specify a 0 (default) to return a file number in the range 1 – 255, inclusive. Specify a 1 to return a file number in the range 256 – 511.

Remarks

Use **FreeFile** to supply a file number that is not already in use.

Example

This example uses the **FreeFile** function to return the next available file number. Five files are opened for output within the loop, and some sample data is written to each.

```
Dim MyIndex, FileNumber
For MyIndex = 1 To 5                        ' Loop 5 times.
    FileNumber = FreeFile                   ' Get unused file number.
    Open "TEST" & MyIndex For Output As #FileNumber ' Create file name.
    Write #FileNumber, "Hello World."       ' Output text
    Close #FileNumber                       ' Close file.
Next MyIndex
```

See Also **Open Statement**

Functions Index

Functions Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

- Abs
- Asc
- Atn
- Bin
- Bin$

- CBool
- CByte
- CDate
- CDbl
- Chr
- Chr$
- CInt

- CLng
- Cos
- CSng
- CStr
- CurDir
- CurDir$
- CVar
- CVDate

- Date
- Dir

- EOF
- Erl
- Err
- Error$
- Exp

- FileAttr
- FileDateTime
- FileLen
- Fix
- Format
- Format$
- FormatDateTime
- FormatNumber
- FreeFile

- GetAttr
- GetOpenFileName
- GetSaveFileName

- Hex
- Hex$

- Input
- Input$
- InputBox
- InputBox$
- InStr
- Int
- IsDate
- IsEmpty
- IsNull
- IsNumeric

- LCase

- LCase$
- Left
- Left$
- Len
- Loc
- LOF
- Log
- LTrim
- LTrim$

- Mid
- Mid$
- MsgBox

- Now

- Oct
- Oct$

- Right

- Right$
- Rnd
- Round
- RTrim
- RTrim$

- Seek
- Shell
- Sin
- Sgn
- Space
- Space$
- Spc
- Str
- Str$
- StrComp
- String
- String$
- Sqr

- Tab
- Tan
- Time
- Timer
- Trim
- Trim$

- UCase
- UCase$

- Val

-
VarType

Functions

Functions

Category	Actions	Functions
Converting	ANSI value to string	Chr, Chr$
	Decimal numbers to other	Bin, Bin$, Hex, Hex$, Oct, Oct$
	Date to string	FormatDateTime
	Number to string	Format, Format$, FormatNumber, Str, Str$
	One numeric data type to another	CBool, CByte, CDbl, CInt, CLng, CSng, CStr, CVar, CDate, CVDate, Fix, Int
	String to ASCII value	Asc
	String to number	Val
Date/time	Get current date or time	Date, Now, Time, Timer
Error trapping	Get error message	Error$
	Get error-status data	Err, Erl
File I/O	Control output appearance	Spc, Tab
	Get information about a file	EOF, FileAttr, FileDateTime, FileLen, FreeFile, Loc, LOF, Seek
	Manage directories	CurDir, CurDir$, Dir

	Read from a file	Input, Input$
	Get file attribute	GetAttr
Math	General calculations	Exp, Log, Sqr
	Generate random numbers	Rnd
	Get absolute value	Abs
	Get the sign of an expression	Sgn
	Numeric conversion	Fix, Int, Round
	Trigonometry	Atn, Cos, Sin, Tan
Strings	Compare two strings	StrComp
	Convert to lowercase or uppercase letters	LCase, LCase$, UCase, UCase$
	Create strings of repeating characters	Space, Space$, String, String$
	Get the length of a string	Len
	Format strings	Format, Format$
	Manipulate strings	InStr, Left, Left$, LTrim, LTrim$, Mid, Mid$, Right, Right$, RTrim, RTrim$, Trim, Trim$
	Work with ASCII and ANSI values	Asc, Chr, Chr$
Variables and constants	Get information about a variable	IsDate, IsEmpty, IsNull, IsNumeric, VarType
Miscellaneous	Run other programs	Shell
	Show input box dialog or message box dialog	InputBox, InputBox$, MsgBox
	Show open/save dialogs	GetOpenFileName, GetSaveFileName
Index	Alphabetical list of functions	

Function...End Function Statement

Function...End Function Statement

Declares the name, arguments, and code that form the body of a **Function** procedure.

Syntax

Function *name* ([*arglist*]) [**As** *type*]
[*statements*]

[*name* = *expression*]
[**Exit Function**]
[*statements*]
[*name* = *expression*]

End Function

The **Function** statement syntax has these Elements:

Element	Description
name	Required. Name of the **Function**; follows standard variable naming conventions.
arglist	Optional. List of variables representing arguments that are passed to the **Function** procedure when it is called. Multiple variables are separated by commas.
type	Optional. Data type of the value returned by the **Function** procedure; may be Byte, Boolean, Integer, Long, Single, Double, Date, String (except fixed length), Object , Variant or an object type.
statements	Optional. Any group of statements to be executed within the **Function** procedure.
expression	Optional. Return value of the **Function**.

The *arglist* argument has the following syntax and Elements:
[**ByVal** | **ByRef**] *varname* [**As** *type*] [=*defval*]

Element	Description
ByVal	Optional. Indicates that the argument is passed by value. **ByVal** is the default in ConceptDraw Basic.
ByRef	Optional. Indicates that the argument is passed by reference.
varname	Required. Name of the variable representing the argument being passed to the procedure; follows standard variable naming conventions.
type	Optional. Data type of the argument passed to the procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Single**, **Double**, **Date**, **String** (variable length only), **Object**, **Variant** or an object type.
defval	Optional. Constant that determine the value that will be passed to the procedure by default if this argument is omitted.

Remarks

Function procedures can be recursive; that is, they can call themselves to perform a given task. However, recursion can lead to stack overflow.

The **Exit Function** statement causes an immediate exit from a **Function** procedure. Program execution continues with the statement following the statement that called the **Function** procedure. Any number of **Exit Function** statements can appear anywhere in a **Function** procedure.

Like a **Sub** procedure, a **Function** procedure is a separate procedure that can take arguments, perform a series of statements, and change the values of its arguments. However, unlike a **Sub** procedure, you can use a **Function** procedure on the right side of an expression in the same way you use any intrinsic function, such as **Sqr**, **Cos**, or **Chr**, when you want to use the value returned by the function.

You call a **Function** procedure using the function name, followed by the argument list in parentheses, in an expression. See the **Call** statement for specific information on how to call **Function** procedures.

To return a value from a function, assign the value to the function name. Any number of such assignments can appear anywhere within the procedure. If no value is assigned to *name*, the procedure returns a default value: a numeric function returns 0, a string function returns a zero-length string (""), and a **Variant** function returns **Empty**. A function that returns an object reference returns **Nothing** if no object reference is assigned to *name* (using **Set**) within the **Function**.

Variables used in **Function** procedures fall into two categories: those that are explicitly declared within the procedure and those that are not. Variables that are explicitly declared in a procedure (using **Dim** or the equivalent) are always local to the procedure. Variables that are used but not explicitly declared in a procedure are also local unless they are explicitly declared at some higher level outside the procedure.

Example

This example uses the **Function** statement to declare the name, arguments, and code that form the body of a **Function** procedure.

```
' The following user-defined function returns the square root of the
' argument passed to it.
Function CalculateSquareRoot(NumberArg As Double) As Double
   If NumberArg < 0 Then    ' Evaluate argument.
      Exit Function    ' Exit to calling procedure.
   Else
      CalculateSquareRoot = Sqr(NumberArg)    ' Returns square root.
   End If
End Function
```

See Also Call Statement , Dim Statement , Exit Statement , Sub Statement

GetAttr Function

GetAttr Function

Returns an **Integer** representing the attributes of a file, directory, or folder.

Syntax

GetAttr(*path*)

Remarks

The required *path* argument is a string expression that specifies a file name. The *path* may include the directory or folder, and the drive.

Return Values

The value returned by **GetAttr** is the sum of the following attribute values:

Constant	**Value**	**Description**
cdbNormal	0	Normal
cdbHidden	2	Hidden
cdbSystem	4	System (Microsoft Windows only)
cdbDirectory	16	Directory or folder
cdbArchive	32	File has changed since last backup (Microsoft Windows only)
cdbAlias	64	File name is an alias (Macintosh only)

Remarks

To determine which attributes are set, use the **And** operator to perform a bitwise comparison of the value returned by the **GetAttr** function and the value of the individual file attribute you want. If the result is not zero, that attribute is set for the named file. For example, the return value of the following **And** expression is zero if the **Archive** attribute is not set:

Result = **GetAttr**(FName) **And** cdbArchive

A nonzero value is returned if the **Archive** attribute is set.

Example

This example uses the **GetAttr** function to determine the attributes of a file and directory or folder.

```
Dim MyAttr
```

```
' Assume file TESTFILE has hidden attribute set.
MyAttr = GetAttr("TESTFILE")  ' Returns 2.
' Returns nonzero  if hidden attribute is set on TESTFILE.
Trace MyAttr And cdbHidden
' Assume file TESTFILE has hidden and read-only attributes set.
MyAttr = GetAttr("TESTFILE")  ' Returns 3.
' Returns nonzero if hidden and read-only attributes are set on TESTFILE.
Trace MyAttr And (cdbHidden + cdbReadOnly)
' Assume MYDIR is a directory or folder.
MyAttr = GetAttr("MYDIR")     ' Returns 16.
```

See Also SetAttr Statement, And Operator , FileAttr Function

GetOpenFileName Function

GetOpenFileName Function

Creates a dialog box, allowing the user to chose a drive, directory and file name and returns a **FixStr** value containing full path to the file.

Syntax

GetOpenFileName([*extention*][,*extentionInfo*][,*title*][,*preview*])

The **GetOpenFileName** function syntax has these arguments:

Element	**Description**
extention	Optional; **Variant** (**String**). File extentions separated with (|).
extentionInfo	Optional; **Variant** (**String**). Describes extentions, specified in *extention*. Elements in *extentionInfo* are separated with (|).
title	Optional. String expression, displayed in the title bar of the dialog box. If you omit *title*, the application name is placed in the title bar.
preview	Optional; **Boolean**. Specifies whether Preview is turned on or off in the dialog box. The default value is **False**.

Example

```
Dim RetVal, Path
Path = GetOpenFileName("exe|any", "Executable Files|Any Format", "MyTitle",
False)
RetVal = Shell(Path, 1)   ' Run a application.
```

See Also GetSaveFileName

GetSaveFileName Function

GetSaveFileName Function

Creates a dialog box, allowing the user to chose a drive, directory and file name and returns a **FixStr** value containing full path to the file.

Syntax
GetSaveFileName([*extention*][,*extentionInfo*][,*title*][,*defaultFile*][,*readonlyFlag*])

The **GetSaveFileName** function syntax has these arguments:

Element	**Description**
extention	Optional; **Variant** (**String**). File extentions separated with (|).
extentionInfo	Optional; **Variant** (**String**). Describes extentions, specified in *extention*. Elements in *extentionInfo* are separated with (|).
title	Optional. String expression, displayed in the title bar of the dialog box. If you omit *title*, the application name is placed in the title bar.
defaultFile	Optional. String expression displayed in the edit line of the dialog box. Specifies a filename suggested by default.
readonlyFlag	Optional; **Boolean**. If the file chosen by the user in the dialog box is read-only, and the *readonlyFlag* is **True**, a warning is displayed and the user is asked to choose another file name. If *readonlyFlag* is **False**, no warning will be displayed and the function will return the path to the specified file. The default value of this argument is **False**.

Example

This example uses **GetSaveFileName** to call a the dialog box and write a string to the selected file.

```
Dim  Path, n, Check
Path = GetSaveFileName("cdb|txt", "CD Basic|Text Format", "MyTitle",
"example1.cdb", True)
Check = StrComp(Path, "")
If Check = 0 Then MsgBox("No file is chosen")
Else
```

```
  n = FreeFile()
  Open Path For Output As #n
  Print #n,"some text"
  Close #n
EndIf
```

See Also GetOpenFileName

Get Statement

Get Statement

Reads data into a variable from an open file on the disk.

Syntax

Get [#]*filenumber*, [*recnumber*], *varname*

The **Get** statement syntax contains the following Elements:

Element	**Description**
filenumber	Requied, any valid file number.
recnumber	Optional, of **Variant** (**Long**) type. Sets the record number (for files in the **Random** mode) or byte number (for files in the **Binary** mode) from which to start reading.
varname	Required, a valid name of the variable in which the read data will be stored.

Remarks

Data, read using the **Get** statement are normally written to a file with the **Put** statement. Number 1 corresponds to the first record (or byte) of the file, number 2 - to the second one, and so on. If the *recnumber* argument is omitted, reading starts from the record (byte) to which the pointer has been moved after the most recent **Get** or **Put** operation (or where it has been moved after the last **Seek** function call). The comma separators are required, for instance:

Get #4,,FileBuffer

The following rules apply to the files, opened in the **Random** mode:

- Even if the lenght of data to be read is less than the lenght of the record, specified in the **Len** parameter of the **Open** statement, the **Get** statement starts reading each subsequent record from the beginning of this record. The space between the end of one record and the start of the following one gets filled with the contents of the file buffer. As it's hard to calculate exactly the amount of data, used for filling, it's recommended that the record lenght be the same as the lenght of data being read.

- If data is read to a string of variable lenght, the **Get** statement first reads the 2-byte descriptor indicating the string lenght, and then the data to be put into the variable. So the record lenght specified in the **Len** parameter of the **Open** statement must be at least 2 byte greater than the actual string lenght.
- If data is read into a **Variant** variable of numeric type, the **Get** statement first reads the 2 bytes indicating the subtype (**VarType**) of this variable, and then the data to be put into this variable. For instance, when reading the **Variant** variable of **VarType** 3 subtype the **Get** statement reads 6 bytes: 2 bytes indicating the subtype of the **Variant** variable as **VarType** 3 (**Long**), and 4 bytes containing the value of the **Long** type. The record lenght specified in the **Len** parameter of the **Open** statement must be at least 2 byte greater than the actual size needed to store this variable.

The above rules apply to files opened in the **Binary** mode, except of the following:

- The **Len** parameter of the **Open** statement is ignored. **Get** reads all variables from the disk continuously - i.e. without filling the space between records with file buffer contents.
- When reading any arrays, except for those which are elements of user-defined types, the **Get** statement reads only data. The descriptor is not read.
- When reading strings of variable lenght which are not elements of user-defined types the 2-byte descriptor is not read. The number of bytes being read is equal to the number of symbols contained in the string. The statements below read 10 bytes from the file with number 1:

VarString = String(10," ")
Get #1,,VarString

Example

In this example the **Get** statement is used for reading data from a file into a variable. It's assumed that the TESTFILE file contains 5 records (see the example of using **Put**).

```
Dim sName As String * 20, nPosition ' Declares variable.
' Opens file for random access.
Open "TESTFILE" For Random As #1 Len = 21
' Reads from the file using the Get statement.
nPosition = 3              ' Determines  record number.
Get #1, nPosition, sName   ' Reads the third record.
MsgBox(sName)
Close #1                   ' Closes file.
```

See Also **Recording data in a file**, **Put Statement**, **Open Statement**, **Seek Function**, **VarType Function**

GoSub...Return Statement

Branches to and returns from a subroutine.

Syntax

GoSub *line*
...

line
...

Return

The *line* argument can be any line label or line number.

Remarks

You can use **GoSub** and **Return** anywhere in a procedure, but **GoSub** and the corresponding **Return** statement must be in the same procedure. A subroutine can contain more than one **Return** statement, but the first **Return** statement encountered causes the flow of execution to branch back to the statement immediately following the most recently executed **GoSub** statement.

Also **GoSub...Return** can be used in global area of visibility.

Note: You can't enter or exit **Sub**/**Function** procedures with **GoSub...Return**.

Tip: Creating separate procedures that you can call may provide a more structured alternative to using **GoSub...Return**.

Example

This example uses **GoSub** to call a subroutine within a **Sub** procedure. The **Return** statement causes the execution to resume at the statement immediately following the **GoSub** statement. The **Exit Sub** statement is used to prevent control from accidentally flowing into the subroutine.

```
Sub GosubDemo()
Dim Num
' Solicit a number from the user.
   Num = InputBox("Enter a positive number to be divided by 2.")
' Only use routine if user enters a positive number.
   If Num > 0 Then
       GoSub MyRoutine
   End If
   Trace Num
   Exit Sub   ' Use Exit to prevent an error.
MyRoutine:
   Num = Num/2   ' Perform the division.
   Return   ' Return control to statement.
```

```
End Sub   ' End of the GosubDemo() Sub.
```

See Also GoTo Statement, On..GoSub Statement, On...GoTo Statement, Sub Statement, Function Statement

GoTo Statement

GoTo Statement

Branches unconditionally to a specified line.

Syntax

GoTo *line*

The required *line* argument can be any line label or line number.

Remarks

GoTo can branch only to lines within the procedure where it appears.

GoTo can also be used in the global area of visibility.

Note: Too many GoTo statements can make code difficult to read and debug. Use structured control statements (**Do...Loop**, **For...Next**, **If...Then...Else**, **Select Case**) whenever possible.

Example

```
Sub GotoDemo()
Dim Number As Double, MyString
   Number = InputBox("Enter number :")     ' Initialize variable.
   ' Evaluate Number and branch to appropriate label.
   If Number = 1 Then
      GoTo Line1
   Else
      GoTo Line2
   End If
Line1:
   MyString = "Number equals 1"
   GoTo LastLine   ' Go to LastLine.
Line2:
   MyString = "Number equals " & Number
LastLine:
   Trace MyString
End Sub
```

See Also Do...Loop Statement, For...Next Statement, GoSub...Return Statement, If...Then...Else Statement, Select Case Statement

Hex Function

Hex Function

Returns a **FixStr** (**String**) value representing the hexadecimal value of a number.

Syntax

Hex[$]([*number*])

The optional *number* argument is any valid numeric expression or string expression in the range from -2147483648 to 2147483647. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 0.

Remarks

If *number* is not already a whole number, it's rounded to the nearest whole number before being evaluated. If *number* is **Empty** or **Null**, the function returns 0. For any other number the **Hex** function returns up to 8 hexadecimal symbols.

You can represent hexadecimal numbers directly by preceding numbers in the proper range with &H. For example, &H10 represents decimal 16 in hexadecimal notation.

The **Hex$** form returns **String** values. The **Hex** form returns **FixStr** values.

Example

```
Dim MyHex
MyHex = Hex(5)    ' Returns 5.
MyHex = Hex(10)    ' Returns A.
MyHex = Hex(459)    ' Returns 1CB.
```

See Also **Bin Function**, **Oct Function**, **Type Conversion Functions**

If...Then...Else Statement

#If...#Else...#Endif Preprocessor Directive

Used to control conditional compilation.

Syntax

#If *TargetBoolean*

[*statements*] //OS-specific code

[#Else

[*elsestatements*]] // Other OS-specific code

#EndIf

The **#If...#Else..#Endif** directive syntax has these Elements:

Element	**Description**
TargetBoolean	Required. **Target_MacOS**, **Target_Win32** constant, used to determine the operating system that will include the code the follows.
statements	Optional. One or more statements that are executed if *TargetBoolean* is **True**.
elsestatements	Optional. One or more statements executed if *TargetBoolean* is **False**.

Remarks

Use conditional compilation to isolate platform-specific statements such as toolbox calls or AppleEvent routines. The code following the **#If** directive is included only in the build for that operating system.

Example

```
Dim Separator as String
#If Target_MacOS
Separator=":"
#Endif
#If Target_Win32
Separator="\"
#Endif
```

See Also

If...Then...Else Statement

Conditionally executes a group of statements, depending on the value of an expression.

Syntax

If *condition* **Then**
[*statements*]

[**ElseIf** *condition-n* **Then**
[*elseifstatements*] ...

[**Else**
[*elsestatements*]]

End If

The **If...Then...Else** statement syntax has these Elements:

Element	**Description**
condition	Required. Expression that is **True** or **False**.
statements	Optional. One or more statements that are executed if *condition* is **True**.
condition-n	Optional. Same as *condition*.
elseifstatements	Optional. One or more statements executed if associated *condition-n* is **True**.
elsestatements	Optional. One or more statements executed if no previous *condition* or *condition-n* expression is **True**.

Remarks

The **If** statement must be the first statement on a line. The **Else**, **ElseIf**, and **End If** Elements of the statement can have only a line number or line label preceding them. The **If** block must end with an **End If** statement.

The **Else** and **ElseIf** clauses are both optional. You can have as many **ElseIf** clauses as you want in a **If** block, but none can appear after an **Else** clause. **If** statements can be nested; that is, contained within one another.

When executing a **If** block, *condition* is tested. If *condition* is **True**, the statements following **Then** are executed. If *condition* is **False**, each **ElseIf** condition (if any) is evaluated in turn. When a **True** condition is found, the statements immediately following the associated **Then** are executed. If none of the **ElseIf** conditions are **True** (or if there are no **ElseIf** clauses), the statements following **Else** are executed. After executing the statements following **Then** or **Else**, execution continues with the statement following **End If**.

Tip Select Case may be more useful when evaluating a single expression that has several possible actions.

Example

This example illustrates the use of the **If...Then...Else** statement.

```
Dim Number As Integer, Digits As Integer
Number = InputBox("Enter a number (0-999):")   ' Initialize variable.
If Number < 10 Then
   Digits = 1
ElseIf Number < 100 Then
   Digits = 2
Else
   Digits = 3
End If
Trace Digits
```

See Also Select Case Statement

IMP Operator

IMP Operator

Used to perform a logical implication on two expressions.

Syntax

result = *expression1* **Imp** *expression2*

The Imp operator syntax has these Elements:

Element	Description
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

The following table illustrates how result is determined:

If expression1 is	And expression2 is	The result is
True	True	True
True	False	False
False	True	True
False	False	True

The Imp operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following table:

If expression1 is	And expression2 is	The result is
0	0	1
0	1	1
1	0	0
1	1	1

Example

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null    ' Initialize variables.
MyCheck = A > B Imp B > C    ' Returns True.
trace MyCheck
MyCheck = A > B Imp C > B    ' Returns False.
trace MyCheck
MyCheck = B > A Imp C > B    ' Returns True.
trace MyCheck
MyCheck = B > A Imp C > D    ' Returns True.
trace MyCheck
MyCheck = C > D Imp B > A    ' Returns False.
trace MyCheck
MyCheck = B Imp A    ' Returns -1 (bitwise comparison).
trace MyCheck
```

See Also [Operators]

InputBox Function

InputBox Function

Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns a **FixStr (String)** containing the contents of the text box.

Syntax

InputBox[$]([*prompt*][, *title*] [, *default*] [, *xpos*] [, *ypos*] [, *helpfile*, *context*])

Element	Description
prompt	Optional. String expression displayed as the message in the dialog box. The maximum length of ***prompt*** is approximately 256 characters, depending on the width of the characters used. If ***prompt*** consists of more than one line, you can separate the lines using a carriage return character (Chr(13)), a linefeed character (**Chr**(10)), or carriage return–linefeed character combination (**Chr**(13) & **Chr**(10)) between each line.
title	Optional. String expression displayed in the title bar of the dialog box. If you omit ***title***, the application name is placed in the title bar.
default	Optional. String expression displayed in the text box as the default response if no other input is provided. If you omit ***default***, the text box is displayed empty.
xpos	Optional. Numeric expression that specifies, in twips, the horizontal distance of the left edge of the dialog box from the left edge of the screen. If ***xpos*** is omitted, the dialog box is horizontally centered.
ypos	Optional. Numeric expression that specifies, in twips, the vertical distance of the upper edge of the dialog box from the top of the screen. If ***ypos*** is omitted, the dialog box is vertically positioned approximately one-third of the way down the screen
helpfile	Optional. String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If ***helpfile*** is provided, ***context*** must also be provided.
context	Optional. Numeric expression that is the Help context number assigned to the appropriate Help topic by the Help author. If ***context*** is provided, ***helpfile*** must also be provided.

The **InputBox$** form returns **String** values. The **InputBox** form returns **FixStr** values.

Remarks

If the user clicks **OK** or presses ENTER , the **InputBox** function returns whatever is in the text box. If the user clicks **Cancel** or presses Esc, the function returns a zero-length string ("").

Example

This example shows various ways to use the **InputBox** function to prompt the user to enter a value. If the x and y positions are omitted, the dialog box is automatically centered for the respective axes. The variable MyValue contains the value entered by the user if the user clicks **OK** or presses the ENTER key . If the user clicks **Cancel**, a zero-length string is returned.

```
Dim Message, Title, Default, MyValue
Message = "Enter a value between 1 and 3"   ' Set prompt.
Title = "InputBox Demo"   ' Set title.
Default = "1"   ' Set default.
' Display message, title, and default value.
MyValue = InputBox(Message, Title, Default)
' Use Helpfile and context. The Help button is added automatically.
MyValue = InputBox(Message, Title, , , , "DEMO.HLP", 10)
' Display dialog box at position 100, 100.
MyValue = InputBox(Message, Title, Default, 100, 100)
```

See Also [MsgBox Function]

Input Function

Input Function

Returns **FixStr** (**String**) containing characters from a file opened in **Input** or **Binary** mode.

Syntax

Input[$](*number*, *filenumber*)

The **Input** function syntax has these Elements:

Element	Description
number	Required. Any valid numeric expression specifying the number of characters to return.
filenumber	Required. Any valid file number.

Remarks

Data read with the **Input** function is usually written to a file with **Print #** or **Put**. Use this function only with files opened in **Input** or **Binary** mode.

Unlike the **Input #** statement, the **Input** function returns all of the characters it reads, including commas, carriage returns, linefeeds, quotation marks, and leading spaces.

With files opened for **Binary** access, an attempt to read through the file using the **Input** function until **EOF** returns **True** generates an error. Use the **LOF** and **Loc** functions instead of **EOF** when reading binary files with **Input**, or use **Get** when using the **EOF** function.

The **Input$** form returns **String** values. The **Input** form returns **FixStr** values.

Example

This example uses the **Input** function to read one character at a time from a file and print it to the Output window.
This example assumes that TESTFILE is a text file with a few lines of sample data.

```
Dim MyChar
Open "TESTFILE" For Input As #1 ' Open file.
Do While Not EOF(1)         ' Loop until end of file.
    MyChar = Input(1, 1)   ' Get one character.
    Trace MyChar            ' Print to the Output window.
Loop
Close #1          ' Close file.
```

See Also

Input # Statement

Input Statement

Input # Statement

Reads data from an open ... file and assigns them to variables.

Syntax

Input #*filenumber*, *varlist*

The **Input** # statement syntax has the following Elements:

Element	**Description**
filenumber	Required, any valid file number.
varlist	Comma-delimited list of variables, to which the values read from the file are to be assigned. These can't be arrays or object variables. However variables describing array elements can be used.

Remarks

Normally data read with the **Input** # statement is written to a file with the help of the **Write #** statement. This statement can only be applied to the files, opened in the **Input** or **Binary** modes.

Standard strings and numeric values that have been input are assigned to variables without any changes. The following table demonstrates how other data is processed:

Data	**Value, assigned to variable**
Comma-separator or an empty string	Empty
#NULL#	NULL
#TRUE# or #FALSE#	True or False
#yyyy-mm-dd hh:mm:ss#	Resulting date and/or time.

Quotation marks (" ") inside data being read are ignored.
Data elements in the file must follow in the same order as the variables in the *varlist* and have data types corresponding to the variables. If a variable is numeric, and the data element is non-numeric, the variable is set to null value.
If end of file is reached when reading a data element, data input stops and an error is generated.

Note: To ensure correct input of file data into variables when using **Input** #, you should always use the **Write #** statement (rather than **Print #**) when writing data into files. Using this statement ensures correct placement of separators between data elements.

Example

In this example the **Input #** statement is used for reading data from a file into two variables. It's assumed that the TESTFILE file exists and contains several text strings, which have been written with the Write # statement - i.e. each string contains a string in quotation marks and a number, separated by a comma, e.g. ("Hello World", 234).

```
Dim MyString
Open "TESTFILE" For Input As #1        ' Opens file for reading.
Do While Not EOF(1)                    ' Loop until the end of file.
    Input #1, MyString                 ' Read data into two variables.
    Trace MyString                     ' Outputs data into the output window.
Loop
Close #1                               ' Closes file.
```

See Also **Recording Data in a File**, **Open Statement**, **Print # Statement**, **Write # Statement**, **Input Function**

InStr Function

InStr Function

Returns a **Long** specifying the position of the first occurrence of one string within another.

Syntax

InStr([*start*,]*string1*, *string2*[, *compare*])

The **InStr** function syntax has these arguments:

Element	Description
start	Optional. Numeric expression that sets the starting position for each search. If omitted, search begins at the first character position. If *start* contains **Null**, an error occurs. The *start* argument is required if *compare* is specified.
string1	Required. String expression being searched.
string2	Required. String expression sought.
compare	Optional. Specifies the type of string comparison. The *compare* may be omitted or have 0 or 1 values. Specify 0 (default) for binary comparison. For textual comparison which is not case-sensitive, specify 1. If *compare* is **Null**, an error occurs.

Remarks

Return Values

If	InStr returns
string1 is zero-length	0
string1 is **Null**	**Null**
string2 is zero-length	*start*
string2 is **Null**	**Null**
string2 is not found	0
string2 is found in *string1*	Position at which match is found
start > *string2*	0

Example

This example uses the **InStr** function to return the position of the first occurrence of one string within another.

```
Dim SearchString, SearchChar, MyPos
SearchString ="XXpXXpXXPXXP"  ' String to search in.
SearchChar = "P"              ' Search for "P".
' A textual comparison starting at position 4. Returns 6.
MyPos = Instr(4, SearchString, SearchChar, 1)
' A binary comparison starting at position 1. Returns 9.
MyPos = Instr(1, SearchString, SearchChar, 0)
' Comparison is binary by default (last argument is omitted).
MyPos = Instr(SearchString, SearchChar)  ' Returns  9.
MyPos = Instr(1, SearchString, "W")      ' Returns 0.
```

See Also StrComp Function

\ Operator

\ Operator

Used to divide two numbers and return an integer result.

Syntax

result = number1 \ number2

The \ operator syntax has these Elements:

Element	**Description**
result	Required; any numeric variable.
number1	Required; any numeric expression.
number2	Required; any numeric expression.

Remarks

Before division is performed, the numeric expressions are rounded to **Byte**, **Integer**, or **Long** expressions.
Usually, the data type of *result* is a **Byte**, **Byte** variant, **Integer**, **Integer** variant, **Long**, or **Long** variant, regardless of whether *result* is a whole number. Any fractional portion is truncated. However, if one expression is **Null** or **Empty**, result is 0.

Example

This example uses the \ operator to perform integer division.

```
Dim MyValue
MyValue = 11 \ 4   ' Returns 2.
trace MyValue
MyValue = 9 \ 3   ' Returns 3.
trace MyValue
MyValue = 100 \ 3   ' Returns 33.
trace MyValue
```

See Also [Operators]

Int,Fix Function

Int,Fix Functions

Returns the integer portion of a number. The returned value has the same data type as the argument.

Syntax

Int([*num*])

Fix([*num*])

The optional *num* argument is a **Double** or any valid numeric expression. If this argument is omitted, is a non-initialized variable, or Null, the function returns 0.

Remarks

Both **Int** and **Fix** remove the fractional Element of number and return the resulting integer value.

The difference between **Int** and **Fix** is that if number is negative, **Int** returns the first negative integer less than or equal to number, whereas **Fix** returns the first negative integer greater than or equal to number. For example, **Int** converts -8.4 to -9, and **Fix** converts -8.4 to -8.

Fix(*num*) is equivalent to:

Sgn(*num*) * **Int**(**Abs**(*num*))

Example

This example illustrates how the **Int** and **Fix** functions return integer portions of numbers. In the case of a negative *num* argument, the Int function returns the first negative integer less than or equal to the *num*; the **Fix** function returns the first negative integer greater than or equal to the *num*.

```
Dim MyNumber
MyNumber = Int(99.8)    ' Returns 99.
MyNumber = Fix(99.2)    ' Returns 99.
MyNumber = Int(-99.8)    ' Returns -100.
MyNumber = Fix(-99.8)    ' Returns -99.
MyNumber = Int(-99.2)    ' Returns -100.
MyNumber = Fix(-99.2)    ' Returns -99.
```

See Also Round Function, Type Conversion Functions

IsDate Function

IsDate Function

Returns a **Boolean** value indicating whether an expression can be converted to a date.

Syntax

IsDate(*expression*)

The required *expression* argument is a date expression or string expression recognizable as a date or time.

Remarks

IsDate returns **True** if the *expression* is a date or is recognizable as a valid date; otherwise, it returns **False**. In Microsoft Windows, the range of valid dates is January 1, 100 A.D. through December 31, 9999 A.D.; the ranges vary among operating systems.

Example

This example uses the **IsDate** function to determine if an expression can be converted to a date.

```
Dim MyDate, YourDate, NoDate, MyCheck
MyDate = "February 12, 1969"
YourDate = #2/12/69#
NoDate = "Hello"
MyCheck = IsDate(MyDate)   ' Returns True.
Trace MyCheck
MyCheck = IsDate(YourDate)   ' Returns True.
Trace MyCheck
MyCheck = IsDate(NoDate)   ' Returns False.
Trace MyCheck
```

See Also IsEmpty Function , IsNull Function , IsNumeric Function , VarType Function

IsEmpty Function

IsEmpty Function

Returns a **Boolean** value indicating whether a variable has been initialized.

Syntax

IsEmpty(*expression*)

The required *expression* argument is a numeric or string expression. However, because **IsEmpty** is used to determine if individual variables are initialized, the *expression* argument is most often a single variable name.

Remarks

IsEmpty returns **True** if the variable is uninitialized, or is explicitly set to Empty; otherwise, it returns **False**. **False** is always returned if *expression* contains more than one variable.

Example

This example uses the **IsEmpty** function to determine whether a variable has been initialized.

```
Dim MyVar, MyCheck
MyCheck = IsEmpty(MyVar)    ' Returns True.
Trace MyCheck
MyVar = Null ' Assign Null.
MyCheck = IsEmpty(MyVar) ' Returns False.
Trace MyCheck
MyVar = Empty ' Assign Empty.
MyCheck = IsEmpty(MyVar) ' Returns True.
Trace MyCheck
```

See Also IsDate Function , IsNull Function , IsNumeric Function , VarType Function

IsNull Function

IsNull Function

Returns a **Boolean** value indicating whether an expression contains no valid data (Null) or can be evaluated to 0.

Syntax

IsNull(*expression*)

The required *expression* argument is a numeric expression , string expression or object variable.

Remarks

IsNull returns **True** if *expression* is **Null**; otherwise, **IsNull** returns **False**.

Null is not the same as Empty, which indicates that a variable has not yet been initialized. It is also not the same as a zero-length string (""), which is sometimes referred to as a null string.

Example

This example uses the **IsNull** function to determine if a variable contains a **Null**.

```
Dim MyVar, MyCheck, MyStr As String
MyCheck = IsNull(MyStr)    ' Returns True.
Trace MyCheck
MyVar = ""
MyCheck = IsNull(MyVar) ' Returns False.
Trace MyCheck
MyVar = Null
MyCheck = IsNull(MyVar) ' Returns True.
Trace MyCheck
```

See Also	IsDate Function , IsEmpty Function , IsNumeric Function , VarType Function

IsNumeric Function

IsNumeric Function

Returns a **Boolean** value indicating whether an expression can be evaluated as a number.

Syntax

IsNumeric(*expression*)

The required *expression* argument is a numeric or string expression.

Remarks

IsNumeric returns **True** if the entire *expression* is recognized as a number; otherwise, it returns **False**.

If *expression* contains a string, the string is evaluated whether it can be converted to a number starting from the beginning until the first non-numeric character.

Example

This example uses the **IsNumeric** function to determine if a variable can be evaluated as a number.

```
Dim MyVar, MyCheck
MyVar = "53"    ' Assign value.
MyCheck = IsNumeric(MyVar)    ' Returns True.
Trace MyCheck
MyVar = "459.95" ' Assign value.
MyCheck = IsNumeric(MyVar) ' Returns True.
Trace MyCheck
MyVar = "Help" ' Assign value.
MyCheck = IsNumeric(MyVar) ' Returns False.
Trace MyCheck
MyVar = 33 ' Assign value.
MyCheck = IsNumeric(MyVar) ' Returns True.
Trace MyCheck
MyVar = "33 la-la-la" ' Assign value.
MyCheck = IsNumeric(MyVar) ' Returns True, because can be evaluated to 33
Trace MyCheck
```

See Also IsDate Function , IsEmpty Function , IsNull Function , VarType Function

IS Operator

IS Operator

Used to compare two object reference variables.

Syntax

result = *object1* Is *object2*

The Is operator syntax has these Elements:

Element	Description

result	Required; any numeric variable.
object1	Required; any object name.
object2	Required; any object name.

Remarks

If *object1* and *object2* both refer to the same object, result is **True**; if they do not, result is **False**. Two variables can be made to refer to the same object in several ways.

In the following example, A has been set to refer to the same object as B:
Set A = B

The following example makes A and B refer to the same object as C
Set A = C
Set B = C

Example

This example uses the Is operator to compare two object references. The object variable names are generic and used for illustration purposes only.

```
Dim MyObject, YourObject, ThisObject, OtherObject, ThatObject, MyCheck
Set MyObject = New DPoint
Set OtherObject = New DPoint
Set YourObject = MyObject   ' Assign object references.
Set ThisObject = MyObject
Set ThatObject = OtherObject
MyCheck = YourObject Is ThisObject   ' Returns True.
trace MyCheck
MyCheck = ThatObject Is ThisObject   ' Returns False.
trace MyCheck
' Assume MyObject <> OtherObject
MyCheck = MyObject Is ThatObject   ' Returns False.
trace MyCheck
```

See Also Operators, Comparison Operators

Keywords

Keywords

Keyword is a word or symbol recognized as Element of the ConceptDraw Basic programming language; for example, a statement, function name, or operator.

Some keywords can be met in several different constructions of the language. These keywords are listed in the table below:

Keyword	**Context of usage**
As	Const Statement, Declare Statement, Dim Statement, Function Statement, Name Statement, Open Statement, ReDim Statement, Static Statement, Sub Statement
ByRef	Declare Statement, Function Statement, Sub Statement
ByVal	Declare Statement, Function Statement, Sub Statement
Date	Date Data Type, Date Function, Date Statement
Else	If...Then...Else Statement, Select Case Statement
Empty	The Empty keyword is used as a Variant subtype. It indicates an uninitialized variable value.
Error	Error Function, Error Statement, On Error Statement
False	The False keyword has a value equal to 0.
For	For...Next Statement, Open Statement
Input	Input Function, Input Statement, Line Input Statement, Open Statement
Is	Is Operator, Select Case Statement
Len	Len Function, Open Statement
Mid	Mid Function, Mid Statement
New	Dim Statement, Set Statement, Static Statement
Next	For...Next Statement, On Error Statement, Resume Statement
Nothing	The Nothing keyword is used to disassociate an object variable from an actual object. Use the Set statement to assign Nothing to an object variable. For example: Set MyObject = Nothing Several object variables can refer to the same actual object. When Nothing is assigned to an object variable, that variable no longer refers to an actual object.
Null	The Null keyword is used as a Variant subtype. It indicates that a variable contains no valid data.
On	On Error Statement, On...GoSub Statement, On...GoTo Statement
Resume	On Error Statement, Resume Statement
Seek	Seek Function, Seek Statement
Static	Function Statement, Static Statement, Sub Statement

String	String Data Type, String Function
Time	Time Function, Time Statement
To	For...Next Statement, Select Case Statement
True	The True keyword has a value equal to 1 for arithmetical operations and -1 for logical operations.

Kill Statement

Kill Statement

Deletes files from the disk.

Syntax
Kill *pathname*

The required *pathname* argument is a string, specifying the names of one or more files to be deleted. The *pathname* argument may include names of directory/folder, or a drive name.

Remarks

An attempt to delete an open file with the **Kill** statement generates an error.

Note: Use the **RmDir** statement to delete directories or folders.

Example

In this example the Kill statement is used to delete a file from disk.

```
' Assume the TESTFILE file exists and is not empty.
Kill "TestFile"   'Deletes the file.
```

See Also	**RmDir Statement**

LCase Function

LCase Function

Returns a **FixStr** (**String**) that has been converted to lowercase.

Syntax
LCase[$](*string*)

The required *string* argument is any valid string expression. If string contains **Null**, **Null** is returned.

Remarks

Only uppercase letters are converted to lowercase; all lowercase letters and nonletter characters remain unchanged.

The **LCase$** form returns **String** values. The **LCase** form returns **FixStr** values.

Example

This example uses the **LCase** function to return a lowercase version of a string.

```
Dim UpperCase, LowerCase
Uppercase = "Hi All 1234"  ' String to convert.
Lowercase = Lcase(UpperCase)    ' returns "hi all 1234".
```

See Also UCase Function

Left Function

Left Function

Returns a **FixStr** (**String**) containing a specified number of characters from the left side of a string.

Syntax
Left(*string*, *length*)

The **Left** function syntax has these named arguments:

Element	**Description**
string	Required. String expression from which the leftmost characters are returned. If *string* contains **Null**, **Null** is returned.

length	Required; **Long**. Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in *string*, the entire string is returned.

Remarks

To determine the number of characters in *string*, use the **Len** function.

The **Left$** form returns **String** values. The **Left** form returns **FixStr** values.

Example

This example uses the **Left** function to return a specified number of characters from the left side of a string.

```
Dim AnyString, MyStr
AnyString = "Hi all"       ' Define string.
MyStr = Left(AnyString, 1)      ' Returns "H".
MyStr = Left(AnyString, 4)      ' Returns "Hi a".
MyStr = Left(AnyString, 20)     ' Returns "Hi all".
```

See Also **Len Function**, **Mid Function** , **Right Function**

Len Function

Len Function

Returns a **Long** containing the number of characters in a string or the number of bytes required to store a variable.

Syntax

Len(*string* | *varname*)

The **Len** function syntax has these Elements:

Element	Description
string	Any valid string expression. If *string* contains **Null**, **Null** is returned.
varname	Any valid variable name. If *varname* contains **Null**, **Null** is returned. If *varname* is a **Variant**, **Len** treats it the same as a **String** and always returns the number of characters it contains.

Remarks

One (and only one) of the two possible arguments must be specified.

Example

```
Dim MyInt As Integer
Dim MyString, MyLen
MyString = "Hi all"  ' Initialize variable.
MyLen = Len(MyInt)        ' Returns 2.
MyLen = Len(MyString)     ' Returns 6.
```

See Also **Data Types**, **InStr Function**

Let Statement

Let Statement

Assigns the value of an expression to a variable or property.

Syntax

[**Let**] *varname* = *expression*

The **Let** statement syntax has these Elements:

Element	**Description**
Let	Optional. Explicit use of the **Let** keyword is a matter of style, but it is usually omitted.
varname	Required. Name of the variable or property; follows standard variable naming conventions.
expression	Required. Value assigned to the variable or property.

Remarks

In ConceptDraw Basic value *expression*, assigned to variable *varname* can be of any type. The type of *expression* will become the same as the type of *varname*.

If *varname* is of **Variant** type, then *varname* will be given the same type as *expression*. It's assumed that the **Variant**-variable *varname* has subtype identical to the type of *expression*.

Use the **Set** statement to assign object references to variables.

Example

This example assigns the values of expressions to variables using the explicit **Let** statement.
```
Dim MyStr, MyInt
' The following variable assignments use the Let statement.
Let MyStr = "Hello World"
Let MyInt = 5
```

The following are the same assignments without the **Let** statement.
```
Dim MyStr, MyInt
MyStr = "Hello World"
MyInt = 5
```

See Also Data Type Summary, Set Statement, Const Statement

Like Operator

Like Operator

Used to compare two strings.

Syntax
result = *string* **Like** *pattern*

The **Like** operator syntax has these Elements:

Element	**Description**
result	Required; any numeric variable.
string	Required; any string expression.
pattern	Required; any string expression conforming to the pattern-matching conventions described in Remarks.

Remarks

If *string* matches pattern, *result* is **True**; if there is no match, *result* is **False**.

In Microsoft Windows the sort order is determined by the code page. In the following example, a typical sort order is shown:

0 < 9 <A < B < E < Z < a < b < e < z

Built-in pattern matching provides a versatile tool for string comparisons. The pattern-matching features allow you to use wildcard characters, character lists, or character ranges, in any combination, to match strings. The following table shows the characters allowed in *pattern* and what they match in *string*:

Characters in *pattern*	**Matches in *string***
?	Any single character.
*****	Zero or more characters.
#	Any single digit (0–9).
[charlist]	Any single character in *charlist*.
[!charlist]	Any single character not in *charlist*.

A group of one or more characters (*charlist*) enclosed in brackets (**[ ]**) can be used to match any single character in *string* and can include almost any character code, including digits.

To match the special characters question mark (**?**), number sign (**#**), and asterisk (*****), enclose them in brackets. The left bracket (**[**) is also a special character, but only if it's followed by the right bracket (**]**). Otherwise it's treated as a regular character. Likewise, right bracket (**]**) in a group is always treated as a special character, but can be used outside a group as an individual character.

By using a hyphen (–) to separate the upper and lower bounds of the range, *charlist* can specify a range of characters. For example, [**A-Z**] results in a match if the corresponding character position in *string* contains any uppercase letters in the range A–Z. Multiple ranges are included within the brackets without delimiters.

Other important rules for pattern matching include the following:

An exclamation point (**!**) at the beginning of *charlist* means that a match is made if any character except the characters in *charlist* is found in *string*. When used outside brackets, the exclamation point matches itself.
A hyphen (–) can appear either at the beginning (after an exclamation point if one is used) or at the end of *charlist* to match itself. In any other location inside the brackets, the hyphen is used to identify a range of characters.

When a range of characters is specified, they may appear in ascending or descending order. The range can be specified by ASCII symbols from number 48 to 122. The only exception is right bracket (**]**), which is number 93 in the ASCII table. For example, [**A-Z**], [**4-1**], [**a-Z**], [**;-Z**] are valid patterns. The expressions in brackets [***-4**], [**Z-.**] [**z-]**] won't be considered as range, but will be treated as individual characters.

Empty brackets **[]** are ignored, they are considered a zero-lenght string ("").

Example

```
Dim Result
Result = "aBBBa" Like "a*a"    ' Returns True.
trace Result
Result = "F" Like "[A-Z]"    ' Returns True.
trace Result
Result = "F" Like "[!A-Z]"    ' Returns False.
trace Result
Result = "a2a" Like "a#a"    ' Returns True.
trace Result
Result = "aM5b" Like "a[L-P]#[!c-e]"    ' Returns True.
trace Result
Result = "BAT123khg" Like "B?T*"    ' Returns True.
trace Result
Result = "CAT123khg" Like "B?T*"    ' Returns False.
trace Result
```

See Also **Operators**, **Comparison Operators**, **InStr Function**, **StrComp Function**

Line Input Statement

Line Input # Statement

Reads a string from an opened file and assigns it to a variable of the String type.

Syntax

Line Input #*filenumber*, *varname*

The **Line Input** # statement syntax has the following Elements:

Element	**Description**
filenumber	Required, any valid file number.
varname	Required, any legal variable name of Variant or String type.

Remarks

Data read with the **Line Input** # statement are normally written to file with the **Print #** statement.

The **Line Input** # statement reads by one symbol at a time until it reaches the carriage return symbol (Chr(13)) or the combination of carriage return and line feed symbols (Chr(13) + Chr(10)). When the string is assigned to the variable, the carriage return and line feed symbols are discarded.

Example

In this example the **Line Input** # statement reads a string from a file and assigns it to a variable. It's assumed that the TESTFILE file exists and contains several lines of text.

```
Dim TextLine
Open "TESTFILE" For Input As #1 ' Opens file.
Do While Not EOF(1)             ' Loop until the end of file.
    Line Input #1, TextLine     'Inputs string into variable.
    Trace TextLine              'Outputs the string into Output window.
Loop
Close #1                        ' Closes file.
```

See Also **Input # Statement**, **Chr Function**

Loc Function

Loc Function

Returns a **Long** specifying the current read/write position within an open file.

Syntax

Loc(*filenumber*)

The required *filenumber* argument is any valid **Integerfile** number.

Remarks

The following describes the return value for each file access mode:

Mode	**Return Value**
Random	Number of the last record read from or written to the file.
Sequential	Current byte position in the file divided by 128. However, information returned by **Loc** for sequential files is neither used nor required.
Binary	Position of the last byte read or written.

Example

This example uses the **Loc** function to return the current read/write position within an open file. This example assumes that TESTFILE is a text file with a few lines of sample data.

```
Dim MyLocation, MyLine
Open "TESTFILE" For Binary As #1    ' Open file just created.
Do While MyLocation < LOF(1)        ' Loop until end of file.
    MyLine = MyLine & Input(1, #1)  ' Read line into variable.
    MyLocation = Loc(1)             ' Get current position within file.
    Trace MyLine                    ' Print to the Output window.
    Trace Tab
    Trace MyLocation
Loop
Close #1                            ' Close file.
```

See Also **Writing Data to a File**, **Seek Statement** , **EOF Function**, **LOF Function**, **Seek Function**

LOF Function

LOF Function

Returns a **Long** representing the size, in bytes, of a file opened using the **Open** statement.

Syntax

LOF(*filenumber*)

The required *filenumber* argument is an **Integer** containing a valid file number.

Remarks

Use the **FileLen** function to obtain the length of a file that is not open.

Example

This example uses the **LOF** function to determine the size of an open file.
This example assumes that TESTFILE is a text file containing sample data.

```
Dim FileLength
Open "TESTFILE" For Input As #1   ' Open file.
FileLength = LOF(1)               ' Get length of file.
Trace FileLength
Close #1                          ' Close file.
```

See Also Open Statement, Loc Function , EOF Function, FileLen Function

Log Function

Log Function

Returns a **Double** specifying the natural logarithm of a number.

Syntax

Log(*num*)

The required *num* argument is a **Double** or any valid numeric expression greater than zero. If this argument is a non-initialized variable, or **Null**, an error occurs.

Remarks

The natural logarithm is the logarithm to the base **e**. The constant **e** is approximately 2.718282.

You can calculate base-*n* logarithms for any number *x* by dividing the natural logarithm of *x* by the natural logarithm of *n* as follows:

Log *n*(*x*) = **Log**(*x*) / **Log**(*n*)

Example

The following example illustrates a custom **Function** that calculates base-10 logarithms:

```
Function Log10(X)
Log10 = Log(X) / Log(10)
End Function
```

See Also Exp Function

LSet Statement

LSet Statement

Justifies the string by the left edge of the string variable.

Syntax

LSet *stringvar = string*

The **LSet** statement syntax has these Elements:

Element	Description
stringvar	Required. The name of the string variable.
string	Required. The string expression to be justified by the left edge of the string variable.

Remarks

The **LSet** function replaces all symbols, remaining in the string variable with spaces.
If the string is longer, than the variable, the **LSet** function copies as many symbols from the start of the string as can fit into the variable.

Example

In this example the LSet statement is used to justify the string by the left edge of the string variable.

```
Dim MyString
MyString = "0123456789"    ' Initializes the string.
Lset MyString = "<-Left"   ' MyString contains "<-Left ".
Trace "|"&MyString&"|"
```

See Also **RSet Statement**, **Data Types**

LTrim Function

LTrim Function

Returns a **FixStr (String)** containing a copy of a specified string without leading spaces.

Syntax

LTrim(*string*)

Remarks

The required *string* argument is any valid string expression. If *string* contains **Null**, **Null** is returned.

The **LTrim$** form returns **String** values. The **LTrim** form returns **FixStr** values.

Example

This example uses the **LTrim** function to strip leading spaces and the **RTrim** function to strip trailing spaces from a string variable. It uses the **Trim** function alone to strip both types of spaces.

```
Dim MyString, TrimString
MyString = " <-Trim-> "              ' Initialize string.
TrimString = LTrim(MyString)         ' TrimString = "<-Trim-> ".
TrimString = RTrim(MyString)         ' TrimString = " <-Trim->".
TrimString = LTrim(RTrim(MyString)) ' TrimString = "<-Trim->".
' Using the Trim function alone achieves the same result.
TrimString = Trim(MyString)          ' TrimString = "<-Trim->".
```

See Also [Left Function](), [Right Function]()

Mid Function

Mid Function

Returns a **FixStr** (**String**) containing a specified number of characters from a string.

Syntax

Mid(*string*, *start*[, *length*])

The **Mid** function syntax has these named arguments:

Element	**Description**
string	Required. String expression from which characters are returned. If string contains **Null**, **Null** is returned.
start	Required; **Long**. Character position in *string* at which the Element to be taken begins. If start is greater than the number of characters in string, **Mid** returns a zero-length string ("").
length	Optional; **Long**. Number of characters to return. If omitted or if there are fewer than *length* characters in the text (including the character at *start*), all characters from the *start* position to the end of the string are returned.

Remarks

To determine the number of characters in *string*, use the **Len** function.

The **Mid$** form returns **String** values. The **Mid** form returns **FixStr** values.

Example

This example uses the **Mid** function to return a specified number of characters from a string.

```
Dim MyString, FirstWord, LastWord, MidWords
MyString = "Mid Function Demo"    ' Create text string.
FirstWord= Mid(MyString, 1, 6)     ' Returns "Mid".
LastWord = Mid(MyString, 16, 3)    ' Returns "Function".
MidWords = Mid(MyString, 8)        ' Returns "Function Demo".
```

See Also [Len Function](), [Left Function](), [Right Function]()

Mid Statement

Mid Statement

Replaces the specified number of symbols in the string of **Variant** (**String**) type with symbols from another string.

Syntax

Mid(*stringvar*, *start*[, *length*]) = *string*

The **Mid** statement syntax has these Elements:

Element	**Description**
stringvar	Required. Name of the string variable to be changed.
start	Required. Value of Variant (Long) type. Defines the position of the symbol in the variable from where to start replacing.
length	Optional. Value of Variant (Long) type. Defiines the number of symbols to be replaced. If this argument is omitted, the entire string will be used.
string	Required. String expression, that serves to replace a Element of string variable.

Remarks

The number of replaced symbols can't exceed the number of symbols in the variable.

Example

This example shows how the Mid statement is used to replace the specified number of symbols of the string variable with symbols from another string.

```
Dim MyString
MyString = "The dog jumps"            ' Initialize string.
Trace MyString
Mid(MyString, 5, 3) = "fox"           ' MyString = "The fox jumps".
Trace MyString
Mid(MyString, 5) = "cow"              ' MyString = "The cow jumps".
Trace MyString
Mid(MyString, 5) = "cow jumped over"  ' MyString = "The cow jumpe".
Trace MyString
Mid(MyString, 5, 3) = "duck"          ' MyString = "The duc jumpe".
Trace MyString
```

See Also Mid Function

- Operator

- Operator

Used to find the difference between two numbers or to indicate the negative value of a numeric expression.

Syntax 1

result = number1 – number2

Syntax 2

–*number*

The - operator syntax has these Elements:

Element	Description
result	Required; any numeric variable.
number	Required; any numeric expression.
number1	Required; any numeric expression.
number2	Required; any numeric expression.

Remarks

In Syntax 1, the – operator is the arithmetic subtraction operator used to find the difference between two numbers.
In Syntax 2, the – operator is used as the unary negation operator to indicate the negative value of an expression.

The data type of result is usually the same as that of the most precise expression. The order of precision, from least to most precise, is **Byte**, **Integer**, **Long**, **Single**, **Double**. The following are exceptions to this order:

If	**Then result is**
The data type of result is a Long, Single, or Date variant that overflows its legal range,	converted to a Variant containing a Double.
The data type of result is a Byte variant that overflows its legal range,	converted to an Integer variant.
The data type of result is an Integer variant that overflows its legal range,	converted to a Long variant.
Subtraction involves a Date and any other data type,	a Date.

One or both expressions are Null or Empty expressions, result is 0.

The order of precision used by addition and subtraction is not the same as the order of precision used by multiplication.

Example

This example uses the - operator to calculate the difference between two numbers.

```
Dim MyResult, Var1
MyResult = 4 - 2   ' Returns 2.
trace MyResult
MyResult = 459.35 - 334.90   ' Returns 124.45.
trace MyResult
Var1 = 5
MyResult = -Var1   ' Returns -5.
trace MyResult
```

See Also Operators

MkDir Statement

MkDir Statement

Creates a new directory or folder.

Syntax

MkDir *path*

Remarks

The required argument *path* is a string specifying the directory or folder to be created. It can contain the drive name. If drive is not specified, **MkDir** created the directory or folder on the current drive.

Example

In this example the MkDir statement is used to create a directory or folder. If the drive is not specified, the new directory/folder will be created on the current drive.

```
MkDir "MYDIR" ' Creates new directory or folder.
```

See Also ChDir Statement, RmDir Statement , CurDir Function

MOD Operator

MOD Operator

Used to divide two numbers and return only the remainder.

Syntax

result = *number1* **Mod** *number2*

The Mod operator syntax has these Elements:

Element	**Description**
result	Required; any numeric variable.
number1	Required; any numeric expression.

number2	Required; any numeric expression.

Remarks

The modulus, or remainder, operator divides *number1* by *number2* (rounding floating-point numbers to integers) and returns only the remainder as *result*.

For example, in the following expression, A (*result*) equals 5.

```
A = 19 Mod 6.7
```

Usually, the data type of result is a **Byte**, **Byte** variant, **Integer**, **Integer** variant, **Long**, or **Variant** containing a **Long**, regardless of whether or not *result* is a whole number. Any fractional portion is truncated.
However, if one expression is **Null** or **Empty** is treated as 0.

Example

This example uses the Mod operator to divide two numbers and return only the remainder. If either number is a floating-point number, it is first rounded to an integer.

```
Dim MyResult
MyResult = 10 Mod 5   ' Returns 0.
trace MyResult
MyResult = 10 Mod 3   ' Returns 1.
trace MyResult
MyResult = 12 Mod 4.3   ' Returns 0.
trace MyResult
MyResult = 12.6 Mod 5   ' Returns 3.
trace MyResult
```

See Also [Operators]

MsgBox Function

MsgBox Function

Displays a message in a dialog box, waits for the user to click a button, and returns an **Integer** indicating which button the user clicked.

Syntax

MsgBox([*prompt*][, *buttons*] [, *title*])

The **MsgBox** function syntax has these named arguments:

Element	Description
prompt	Optional. String expression displayed as the message in the dialog box. The maximum length of *prompt* is approximately 1024 characters, depending on the width of the characters used. If *prompt* consists of more than one line, you can separate the lines using a carriage return character (**Chr**(13)), a linefeed character (**Chr**(10)), or carriage return – linefeed character combination (**Chr**(13) & **Chr**(10)) between each line.
buttons	Optional. Numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If omitted, the default value for *buttons* is 0.
title	Optional. String expression displayed in the title bar of the dialog box. If you omit *title*, the application name is placed in the title bar.

Settings

The *buttons* argument settings are:

Constant	Value	Description
cdbOKOnly	0	Display **OK** button only.
cdbOKCancel	1	Display **OK** and **Cancel** buttons.
cdbAbortRetryIgnore	2	Display **Abort**, **Retry**, and **Ignore** buttons.
cdbYesNoCancel	3	Display **Yes**, **No**, and **Cancel** buttons.
cdbYesNo	4	Display **Yes** and **No** buttons.
cdbRetryCancel	5	Display **Retry** and **Cancel** buttons.
cdbCritical	16	Display **Critical Message** icon.
cdbQuestion	32	Display **Warning Query** icon.
cdbExclamation	48	Display **Warning Message** icon.
cdbInformation	64	Display **Information Message** icon.
cdbDefaultButton1	0	First button is default.
cdbDefaultButton2	256	Second button is default.
cdbDefaultButton3	512	Third button is default.

The first group of values (0–5) describes the number and type of buttons displayed in the dialog box; the second group (16, 32, 48, 64) describes the icon style; the third group (0, 256, 512) determines which button is the default. When adding numbers to create a final value for the *buttons* argument, use only one number from each group.

These constants are specified by ConceptDraw Basic for applications. As a result, the names can be used anywhere in your code in place of the actual values.

Return Values

Constant	Value	Description
cdbOK	1	**OK**
cdbCancel	2	**Cancel**
cdbAbort	3	**Abort**
cdbRetry	4	**Retry**
cdbIgnore	5	**Ignore**
cdbYes	6	**Yes**
cdbNo	7	**No**

Remarks

If the dialog box displays a **Cancel** button, pressing the ESC key has the same effect as clicking **Cancel**.

If some arguments are omitted, you must include the corresponding comma delimiters.

Example

This example uses the **MsgBox** function to display a critical-error message in a dialog box with **Yes** and **No** buttons. The **No** button is specified as the default response. The value returned by the **MsgBox** function depends on the button chosen by the user.

```
Dim Msg, Style, Title, Response, MyString
Msg = "Do you want to continue ?"   ' Define message.
Style = cdbYesNo + cdbCritical + cdbDefaultButton2   ' Define buttons.
Title = "MsgBox Demonstration"   ' Define title.
Response = MsgBox(Msg, Style, Title)
If Response = cdbYes Then   ' User chose Yes.
   MyString = "Yes"   ' Perform some action.
Else   ' User chose No.
   MyString = "No"   ' Perform some action.
End If
```

See Also InputBox Function

** Operator*

* Operator

Used to multiply two numbers.

Syntax

result = expression1 ***** *expression2*

The ***** operator syntax has these Elements:

Element	Description
result	Required; any numeric variable.
expression1	Required; any numeric expression.
expression2	Required; any numeric expression.

Remarks

The data type of result is usually the same as that of the most precise expression. The order of precision, from least to most precise, is **Byte**, **Integer**, **Long**, **Single**, **Double**. The following are exceptions to this order:

If	Then result is
The data type of result is a **Long**, **Single**, or **Date** variant that overflows its legal range,	converted to a Variant containing a **Double**.
The data type of result is a **Byte** variant that overflows its legal range,	converted to an **Integer** variant.
The data type of result is an **Integer** variant that overflows its legal range,	converted to a **Long** variant.

If one or both expressions are **Null** or **Empty** expressions, result is 0.
The order of precision used by multiplication is not the same as the order of precision used by addition and subtraction.

Example

This example uses the * operator to multiply two numbers.

```
Dim MyValue
MyValue = 2 * 2 ' Returns 4.
trace MyValue
MyValue = 459.35 * 334.90 ' Returns 153836.315.
trace MyValue
```

See Also [Operators]

Name Statement

Name Statement

Renames a file, directory or folder.

Syntax

Name *oldpathname* **As** *newpathname*

The **Name** statement syntax has the following Elements:

Element	**Description**
oldpathname	Required. A string specifying the name and path to an existing file. It may include folder and drive names.
newpathname	Required. A string specifying the new file name and path. It may include folder and drive names. A file with such name must not exist.

Remarks

Both arguments, *oldpathname* and *newpathname*, should point to the same drive. If path specified in *newpathname* exists and is different to the path in *oldpathname*, the **Name** statement will move the file into the new directory or folder and rename it (if needed). If the paths in *newpathname* and *oldpathname* are different but the filenames are the same, the **Name** statement will move the file to the new directory or folder without renaming it. With the **Name** statement you can move a file from one directory to another, but you can't move a directory or a folder.

An attempt to renamed an open file using **Name** generates an error. You should close the file prior to renaming it. You can't use wildcards such as (*) or (?) in arguments of the **Name** statement.

Example

In this example the Name statement is used to rename a file. Assume that the specified files and folders exist.

```
Dim OldName, NewName
OldName = "OLDFILE"
NewName = "NEWFILE"           ' Specifies filenames.
Name OldName As NewName       ' Renames the file.
' In Microsoft Windows:
OldName = "C:\MYDIR\OLDFILE"
NewName = "C:\YOURDIR\NEWFILE"
Name OldName As NewName       ' Moves and renames file.
' On the  Macintosh:
OldName = "HD:MY FOLDER:OLDFILE"
NewName = "HD:YOUR FOLDER:NEWFILE"
Name OldName As NewName       ' Moves and renames file.
```

See Also Kill Statement

NOT Operator

NOT Operator

Used to perform logical negation on an expression.

Syntax

result = ***Not*** *expression*

The Not operator syntax has these Elements:

Element	**Description**
result	Required; any numeric variable.
expression	Required; any expression.

Remarks

The following table illustrates how result is determined:

If expression is	**Then result is**
True	False
False	True

In addition, the Not operator inverts the bit values of any variable and sets the corresponding bit in result according to the following table:

If bit in expression is	Then bit in result is
0	1
1	0

Example

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null    ' Initialize variables.
MyCheck = Not(A > B)    ' Returns False.
trace MyCheck
MyCheck = Not(B > A)    ' Returns True.
trace MyCheck
MyCheck = Not(C > D)    ' Returns False.
trace MyCheck
MyCheck = Not A    ' Returns -11 (bitwise comparison).
trace MyCheck
```

See Also [Operators]

Now Function

Now Function

Returns a **Date** value specifying the current date and time according your computer's system date and time.

Syntax

Now()

Example

```
Dim Today
Today = Now()     ' Assign current system date and time
```

See Also Date Function, Date Statement, Time Function, Time Statement

Oct Function

Oct Function

Returns a **FixStr** (**String**) value representing the octal value of a number.

Syntax

Oct[$]([*number*])

The optional *number* argument is any valid numeric expression or string expression in the range from -2147483648 to 2147483647. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 0.

Remarks

If *number* is not already a whole number, it's rounded to the nearest whole number before being evaluated. If *number* is **Empty** or **Null**, the function returns 0. For any other number the **Hex** function returns up to 11 octal symbols.

You can represent octal numbers directly by preceding numbers in the proper range with &O. For example, &O10 represents decimal 8 in octal notation.

The **Oct$** form returns **String** values. The **Oct** form returns **FixStr** values.

Example

```
Dim MyOct
MyOct = Oct(4)     ' Returns 4.
MyOct = Oct(8)    ' Returns 10.
MyOct = Oct(459)   ' Returns 713.
```

See Also [Bin Function](), [Hex Function](), [Type Conversion Functions]()

On Error Statement

On Error Statement

Enables an error-handling routine and specifies the location of the routine; can also be used to disable an error-handling routine.

Syntax

On Error GoTo *line*

On Error Resume Next

On Error GoTo 0

The **On Error** statement syntax can have any of the following forms:

Statement	**Description**
On Error GoTo *line*	Enables the error-handling routine that starts at line specified in the required *line* argument. The *line* argument is any line label or line number. If a run-time error occurs, control branches to *line*, making the error handler active. The specified *line* must be in the same procedure as the **On Error** statement; otherwise, a compile-time error occurs. If the **On Error** statement is in the local area of visibility, the specified specified *line* should be defined in the same area.
On Error Resume Next	Specifies that when a run-time error occurs, control goes to the statement immediately following the statement where the error occurred where execution continues. Use this form rather than **On Error GoTo** when accessing objects.
On Error GoTo 0	Disables any enabled error handler in the current procedure.

Remarks

If you don't use an **On Error** statement, any run-time error that occurs is fatal; that is, an error message is displayed and execution stops.

An "enabled" error handler is one that is turned on by an **On Error** statement; an "active" error handler is an enabled handler that is in the process of handling an error. If an error occurs while an error handler is active (between the occurrence of the error and a **Resume**, **Exit Sub** or **Exit Function** statement), the current procedure's error handler can't handle the error. Control returns to the calling procedure. If the calling procedure has an enabled error handler, it is activated to handle the error. If the calling procedure's error handler is also active, control passes back through previous calling procedures until an enabled, but inactive, error handler is found. If no inactive, enabled error handler is found, the error is fatal at the point at which it actually occurred. Each time the error handler passes control back to a calling procedure, that procedure becomes the current procedure. Once an error is handled by an error handler in any procedure, execution resumes in the current procedure at the point designated by the **Resume** statement.

Note: An error-handling routine is not a **Sub** procedure or **Function** procedure. It is a section of code marked by a line label or line number.

To determine the cause of an error error-handling routines use the value returned by the **Err()** function. Error-handling routines should check or save the values returned by **Err()** and **Erl()** before a new error may occur, or prior to calling a procedure, which may cause an error. These values describe respectively the number of the last error and the line number in the source module. Text of the error message corresponding to the error code can be obtained by using the **Error$()** function.

On **Error Resume Next** causes execution to continue with the statement immediately following the statement that caused the run-time error, or with the statement immediately following the most recent call out of the procedure containing the **On Error Resume Next** statement. This statement allows execution to continue despite a run-time error. You can place the error-handling routine where the error would occur, rather than transferring control to another location within the procedure. An **On Error Resume Next** statement becomes inactive when another procedure is called, so you should execute an **On Error Resume Next** statement in each called routine if you want inline error handling within that routine.

Note: The **On Error Resume Next** construction may be preferable to **On Error GoTo** when handling errors generated during access to other objects.

On Error GoTo 0 disables error handling in the current procedure. It doesn't specify line 0 as the start of the error-handling code, even if the procedure contains a line numbered 0. Without an **On Error GoTo 0** statement, an error handler is automatically disabled when a procedure is exited.

To prevent error-handling code from running when no error has occurred, place an **Exit Sub** or **Exit Function** statement immediately before the error-handling routine, as in the following fragment:

```
Sub foo()
On Error GoTo ErrorHandler
. . .
Exit Sub
ErrorHandler:
. . .
Resume Next
End Sub
```

Here, the error-handling code follows the **Exit Sub** statement and precedes the **End Sub** statement to separate it from the procedure flow. Error-handling code can be placed anywhere in a procedure.

Note: System errors during calls to dynamic-link libraries (DLL) do not raise exceptions and cannot be trapped with ConceptDraw Basic error trapping. When calling DLL functions, you should check each return value for success or failure (according to the API specifications).

Example

```
Sub OnErrorStatementDemo()
   On Error GoTo ErrorHandler   ' Enable error-handling routine.
   Open "TESTFILE" For Output As #1   ' Open file for output.
   Kill "TESTFILE"    ' Attempt to delete open file.
   On Error Goto 0    ' Turn off error trapping.
   On Error Resume Next    ' Defer error trapping.
   Dim d As Double
   d = 10 / sin(0)    ' "Division by zero" error and resume next statement
   d = 20 / cos(0)
   Trace d
Exit Sub       ' Exit to avoid handler.
ErrorHandler:    ' Error-handling routine.
   Select Case Err()    ' Evaluate error number.
      Case 55, 75    ' "File already open" or "Path/File access error" error.
         Trace """File already open"" or ""Path/File access error"" error"
         Close #1    ' Close open file.
      Case Else
         ' Handle other situations here...
         Resume Next
   End Select
   Resume    ' Resume execution at same line that caused the error.
End Sub
```

See Also End Statement, Err Function, Erl Function, Exit Function Statement, Exit Sub Statement, Resume Statement , Trappable errors

On...GoSub Statement

On...GoSub Statement

Branch to one of several specified subroutines , depending on the value of an expression.

Syntax

On *expression* **GoSub** *destinationlist*

The **On...GoSub** statement syntax has these Elements:

Element	**Description**
expression	Required. Any numeric *expression* that evaluates to a whole number between 0 and 255, inclusive. If *expression* is any number other than a whole number, it is rounded before it is evaluated.
destinationlist	Required. List of line numbers or line labels separated by commas.

Remarks

The value of *expression* determines which line is branched to in *destinationlist*. If the value of *expression* is less than 1 or greater than the number of items in the list or greater than 255 then control drops to the statement following **On...GoSub**.

You can mix line numbers and line labels in the same list.

Tip: **Select Case** provides a more structured and flexible way to perform multiple branching.

Example

```
Sub OnGosubDemo()
Dim Number, MyString
   MyString = "Nothing"
   Number = InputBox("Enter branch number:")    ' Initialize variable.
   ' Branch to Sub<Number>.
   On Number GoSub Sub1, Sub2
   Trace MyString
   Exit Sub
Sub1:
   MyString = "In Sub1" : Return
Sub2:
   MyString = "In Sub2" : Return
End Sub
```

See Also	GoSub...Return Statement, GoTo Statement, On...GoTo Statement, Select Case Statement

On...GoTo Statement

On...GoTo Statement

Branch to one of several specified lines, depending on the value of an expression.

Syntax

On *expression* **GoTo** *destinationlist*

The **On...GoTo** statement syntax has these Elements:

Element	Description
expression	Required. Any numeric *expression* that evaluates to a whole number between 0 and 255, inclusive. If *expression* is any number other than a whole number, it is rounded before it is evaluated.

destinationlist	Required. List of line numbers or line labels separated by commas.

Remarks

The value of *expression* determines which line is branched to in *destinationlist*. If the value of *expression* is less than 1 or greater than the number of items in the list or greater than 255 then control drops to the statement following **On...GoTo**.

You can mix line numbers and line labels in the same list.

Tip: **Select Case** provides a more structured and flexible way to perform multiple branching.

Example

```
Sub OnGoToDemo()
Dim Number, MyString
   MyString = "Nothing"
   Number = InputBox("Enter branch number:")    ' Initialize variable.
   ' Branch to Line<Number>.
   On Number GoTo Line1, Line2
TraceHandle:
   Trace MyString
   Exit Sub
Line1:
   MyString = "In Line1" : GoTo TraceHandle
Line2:
   MyString = "In Line2" : GoTo TraceHandle
End Sub
```

See Also	GoSub...Return Statement, GoTo Statement, On...GoSub Statement, Select Case Statement

Open Statement

Open Statement

Allows Input/Output operations with a file.

Syntax

Open *pathname* **For** *mode* [**Access** *access*] [*lock*] **As** [#]*filenumber* [**Len**=*reclength*]

The **Open** statement syntax contains the following Elements:

Element	**Description**

pathname	Required. String expression indicating the filename. The path can contain directory and drive name.
mode	Required. Keyword indicating the file open mode: **Append**, **Binary**, **Input**, **Output** or **Random**. By default, a file is opened in the **Random** access mode.
access	Optional. Keyword specifying operations allowed with the opened file: **Read**, **Write** or **Read Write**.
lock	Optional. Keyword specifying operations that other processes can perform on the opened file: **Shared**, **Lock Read**, **Lock Write** and **Lock Read Write**.
filenumber	Required. File number may range from 1 to 511 inclusive. To find next free file number use the **FreeFile** function.
reclength	Optional. Number less or equal 32 767 (bytes). For files opened in the **Random** mode this value is the lenght of record. For files with serial access this value is the number of symbols read into the buffer.

Remarks

A file must be open in order to perform input/output operations. The **Open** statement reserves the input/output buffer for the file and sets the buffer usage mode.
If the path argument describes a file that doesn't exist, such file will be created when opening in **Append**, **Binary**, **Output** or **Random** modes.
If the file is already opened by some other process and the specified access mode is not allowed, the **Open** statement will not be executed and an error will be generated.

If the *mode* argument is set to **Binary**, the **Len** parameter is ignored.

Example

This example shows different ways of using the **Open** statement for file input/output operations.

Opening TESTFILE for reading.
```
Open "TESTFILE" For Input As #1
' Close file before re-opening in another mode.
Close #1
```

Opening the file in the Binary mode for writing only.
```
Open "TESTFILE" For Binary Access Write As #1
' Close file before re-opening in another mode.
Close #1
```

The following commands open the file for ... output (serial output); any process can read from or write to the file.
```
Open "TESTFILE" For Output Shared As #1
' Close file before re-opening in another mode.
Close #1
```

The following commands open the file in the Binary mode for reading; other processes can't read from this file.

```
Open "TESTFILE" For Binary Access Read Lock Read As #1
' Close the file.
Close #1
```

See Also Recording Data in a File, Close Statement , FreeFile Function

Operators

Operators

This section describes operators in ConceptDraw Basic and their precedence in complex expressions.

When several operations occur in an expression, each Element is evaluated and resolved in a predetermined order. That order is known as operator precedence. Parentheses can be used to override the order of precedence and force some Elements of an expression to be evaluated before others. Operations within parentheses are always performed before those outside. Within parentheses, however, normal operator precedence is maintained.

The operators, supported in ConceptDraw Basic, can be divided into 3 groups:arithmetic, comparison, logical. When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last. Within individual categories, operators are evaluated in the order of precedence shown below:

Arithmetic	**Comparison**	**Logical**
Exponentiation (^ or **)	Equality (=)	NOT
Negation (-)	Inequality (<>)	AND
Multiplication and division (*, /)	Less than (<)	OR
Integer division (\)	Greater than (>)	XOR
Modulo arithmetic (MOD)	Less than or Equal to (<=)	EQV
Adding and substraction (+, -)	Greater than or Equal to (>=)	IMP
String concatenation (&)	LIKE	
Getting address (ADDRESSOF)	IS	

Note: All comparison operators have equal precedence - that is, they are evaluated in the left-to-right order in which they appear.

When multiplication and division occur together in an expression, each operation is evaluated as it occurs from left to right. Likewise, when addition and subtraction occur together in an expression, each operation is evaluated in order of appearance from left to right.

The string concatenation operator (**&**) is not really an arithmetic operator, but in precedence it does fall after all arithmetic operators and before all comparison operators. Similarly, the **Like** operator, while equal in precedence to all comparison operators, is actually a pattern-matching operator.

OR Operator

OR Operator

Used to perform a logical disjunction on two expressions.

Syntax

result = *expression1* **Or** *expression2*

The Or operator syntax has these Elements:

Element	**Description**
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If either or both expressions evaluate to True, result is True. The following table illustrates how result is determined:

If expression1 is	**And expression2 is**	**The result is**
True	True	True
True	False	True
False	True	True
False	False	False

The Or operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following table:

If expression1 is	**And expression2 is**	**The result is**

0	0	0
0	1	1
1	0	1
1	1	1

Example

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null    ' Initialize variables.
MyCheck = A > B Or B > C    ' Returns True.
trace MyCheck
MyCheck = B > A Or B > C    ' Returns True.
trace MyCheck
MyCheck = A > B Or B > D    ' Returns True.
trace MyCheck
MyCheck = B > D Or B > A    ' Returns True.
trace MyCheck
MyCheck = A Or B    ' Returns 10 (bitwise comparison).
trace MyCheck
```

See Also [Operators]

Pause Statement

Pause Statement

Suspends the execution of the script for a specified interval.

Syntax

Pause*TimeoutMilliseconds*

The *TimeoutMilliseconds* parameter specifies the time, in milliseconds, for which to suspend execution.

Remarks

The **Pause** statement is fully equivalent to the **Wait** statement. These two statements work absolutely identically and are supported for compatibility with different versions of BASIC.

Example

In the example below **Pause** is used to suspend execution for 5 seconds.

```
Pause 5000
```

See Also Wait Statement, Timer Function

Print Statement

Print # Statement

Writes display-formatted data to a sequential file.

Syntax

Print #*filenumber*, [*outputlist*]

The **Print** # statement syntax has the following Elements:

Element	**Description**
filenumber	Required. Any valid file number.
outputlist	Optional. Expression or list of expressions to print.

Settings

Below are the valid *outputlist* argument settings:

[{**Spc**(*n*) \| **Tab**[(*n*)]}] [*expression*] [*charpos*]

Setting	**Description**
Spc(*n*)	Used to insert space characters in the output, where *n* is the number of space characters to insert.
Tab(*n*)	Used to position the insertion point to an absolute column number, where *n* is the column number. Use **Tab** with no argument to position the insertion point at the beginning of the next print zone.
expression	Numeric expressions or string expressions to print.
charpos	Specifies the insertion point for the next character. Use a semicolon to position the insertion point immediately after the last character displayed.

	Use **Tab*(n)*** to position the insertion point to an absolute column number. Use **Tab** with no argument to position the insertion point at the beginning of the next print zone. If *charpos* is omitted, the next character is printed on the next line.

Remarks

Data written with **Print** # is usually read from a file with **Line Input** # or **Input**.
If you omit *outputlist* and include only a list separator after *filenumber*, a blank line is printed to the file. Multiple expressions can be separated with either a space or a semicolon. A space has the same effect as a semicolon.
For **Boolean** data, either *True* or *False* is printed. The **True** and **False** keywords are not translated, regardless of the locale settings.

Date data is written to the file using the standard short date format recognized by your system. When either the date or time component is missing or zero, only the Element provided gets written to the file.
Nothing is written to the file if *outputlist* data is **Empty**. However, if *outputlist* data is **Null**, the **Null** keyword is written to file.

All data written to the file using **Print** # is internationally aware; that is, the data is properly formatted using the appropriate decimal separator.
Because **Print** # writes an image of the data to the file, you must delimit the data so it prints correctly. If you use **Tab** with no arguments to move the print position to the next print zone, **Print** # also writes spaces between print fields to the file.

Note: If, at some future time, you want to read the data from a file using the **Input #** statement, use the **Write #** statement instead of the **Print #** statement to write the data to the file. Using **Write #** ensures the integrity of each separate data field by properly delimiting it, so it can be read back in using the **Input #** statement. Using **Write #** also ensures it can be correctly read in any locale.

Example

In this example the **Print** # statement is used to write data to a file.

```
Open "TESTFILE" For Output As #1  ' Opens file for writing.
Print #1, "Example"               ' Prints text to file.
Print #1,                         ' Prints a blank line to file.
Print #1, "Zone 1"; Tab; "Zone 2" ' Prints in two print zones.
Print #1, "Example"; " "; "for all"   ' Lines are separated with a space.
Print #1, Spc(5); "5 Space"     ' Prints five spaces.
Print #1, Tab(10); "Hello"        ' Prints a word in column 10.
' Assigns Boolean, Date values.
Dim MyBool, MyDate, MyNull
MyBool = False
MyDate = #02/12/1969#
MyNull = NULL
' Instead of the True and False their corresponding translations in
' the current language are written.  Date is written
' it the short system format.
Print #1, MyBool ; " - Boolean"
Print #1, MyDate ; " - Date"
Print #1, MyNull ; " - NULL"
Close #1
```

See Also **Writing Data in a File**, **Open Statement** , **Write # Statement**, **Spc Function** , **Tab Function**

Put Statement

Put Statement

Writes data from a variable to a disk file.

Syntax

Put [#]*filenumber*, [*recnumber*], *varname*

The **Put** statement syntax has the following Elements:

Element	**Description**
filenumber	Required. Any valid file number.
recnumber	Optional. Record number (**Random** mode files) or byte number (**Binary** mode files) at which writing begins.
varname	Required. Name of variable containing data to be written to disk.

Remarks

Data written with **Put** is usually read from a file with **Get**.
The first record or byte in a file is at position 1, the second record or file is at position 2 and so on.
If you omit *recnumber*, the next record or byte after the last **Get** or **Put** statement or pointed to by the **Seek** function is written. You must include delimiting commas, for example:

Put #4,,FileBuffer

For files opened in **Random** mode, the following rules apply:

- If the length of the data being written is less than the lenght specified in the **Len** clause of the **Open** statement, **Put** writes subsequent records on record-length boundaries. The space between the end of one record and the beginning of the next record is padded with the existing contents of the file buffer. Because the amount of padded data can't be determined with any certainty, it is generally a good idea to have the record lenght match the length of the data being written. If the length of the data being written is greater than the lenght specified in the **Len** clause in the **Open** statement, an error occurs.

- If the variable being written is a variable-lenght string, **Put** writes a 2-byte descriptor containing the string lenght, and then the variable. The record lenght specified in the **Len** clause in the **Open** statement must be at least 2 bytes greater than the actual lenght of the string.
- If the variable being written is a **Variant** of a numeric type, **Put** writes 2 bytes identifuing the **VarType** of the **Variant**, and then writes the variable. For example, when writing a **Variant VarType** 3, **Put** writes 6 bytes: 2 bytes identyfying the **Variant** as **VarType** 3 (**Long**), and 4 bytes containing the **Long** data. The record lenght specified in the **Len** clause in the **Open** statement must be at least 2 bytes greater than the actual number of bytes required to store the variable.

Example

In this example the Put statement is used to write data to a file.

```
Dim sName as String*20, nRecordNumber   ' Declares variable.
' Opens file for Random access.
Open "TESTFILE" For Random As #1 Len = 21
For nRecordNumber = 1 To 5              '  Repeats the  loop 5 times.
    sName = "My Name " & nRecordNumber  ' Creates a string.
    Put #1, nRecordNumber, sName        ' Writes record to file.
Next nRecordNumber
Close #1                                ' Closes file.
```

See Also [Writing Data to a File](), [Get Statement](), [Open Statement](), [Seek Statement](), [VarType Function]()

Randomize Statement

Randomize Statement

Initializes the random-number generator.

Syntax

Randomize [*number*]

The optional *number* argument is a Variant or any valid numeric expression.

Remarks

Randomize uses *number* to initialize the **Rnd** function's random-number generator, giving it a new seed value. If you omit number, the value returned by the system timer is used as the new seed value.

If **Randomize** is not used, the **Rnd** function (with no arguments) uses the same number as a seed the first time it is called, and thereafter uses the last generated number as a seed value.

Example

This example uses the **Randomize** statement to initialize the random-number generator. Because the number argument has been omitted, **Randomize** uses the return value returned by the system timer as the new seed value.

```
Dim MyValue
Randomize   ' Initialize random-number generator.
MyValue = CInt((6 * Rnd()) + 1) ' Generate random value between 1 and 6.
Trace MyValue
```

See Also Rnd Function , Timer Function

Recording data in a file

Recording data in a file

At work to a large number of data often happens conveniently to write down data in the file or to read out from the file. The instruction of Open allows to create directly the file and to get to it access. The instruction of Open provides three types of access to files:

- Consecutive access (the Input, Output and Append modes modes), usually used for record of text files, for example protocols of mistakes or reports.
- The direct access (Random mode) used if necessary to consider and write down data in the file without its closing. Files of direct access contain data in the form of records which simplify and accelerate search of the necessary data.
- Binary access (Binary mode), is used, when it is required to consider or write down byte in any position in the file, for example at preservation or display of dot images.

Note. The instruction of Open shouldn't be used for access to own types of files of appendices. For example, it is not necessary to use Open for access to the Word document, to a spreadsheet of Microsoft Excel or to the Microsoft Access database as it will cause loss of integrity and file damage. In the following table the instructions which are usually used for data recording in files and for data reading from files are shown.

Access type	Data recording	Data reading
Consecutive	Print #, Write #	Input # , Line Input #
Any	Put	Get
Binary	Put	Get

See Also	Get Statement, Input # Statement, Line Input #, Open Statement, Print # Statement, Put Statement, Write # Statement, Data type summary

ReDim Statement

ReDim Statement

Redeclares variables and reallocate storage space.

Syntax

ReDim [**Preserve**] *varname*[**(***subscripts***)**] [**As** *type*] [**,** *varname*[**(***subscripts***)**] [**As** *type*]] ...

The **ReDim** statement syntax has these Elements:

Element	Description
Preserve	Optional. Keyword used to preserve the data in an existing array when you change the size of the last dimension.
varname	Required. Name of the variable; follows standard variable naming conventions.
subscripts	Optional. Dimensions of an array variable; up to 10 multiple dimensions may be declared. The subscripts argument uses the following syntax: *count1*[, *count2*] . . . where *count1*, *count2* are constants, indicating the upper limit of allowable indices for the defined array. The lower limit of allowable indices always equals 0. So, for a one-dimensional array the number of elements can be calculated as *count1*+1 .
type	Optional. Data type of the variable; may be Byte, Boolean, Integer, Long, Single, Double, Date, String (for variable-length strings), **String** * *length* (for fixed-length strings), Object, Variant, or an object type. Use a separate **As** *type* clause for each variable you declare.

Remarks

The **ReDim** statement is used to declare or resize a dynamic array, which was already described with the **Dim** statement. Also the **ReDim** statement allows to completely redeclare a variable, declared earlier.

It possible to use the **ReDim** statement again to change the number of elements and dimensions of the array.

The **Preserve** keyword can only be used with arrays. It's only possible to modify the last dimension of the array, however it's not possible to change the number of dimensions. For instance, if an array has only one dimension, it's possible to change this dimension as it's the last and only in the array. However, if an array has two or more dimensions, it's only possible to change the value of the last dumension; the contents of the arrays will be preserved. The following example demonstrates how to increase the value of the last dimension of a dynamic array without destroying data it contains.

ReDim A(10, 10, 10)
. . .
ReDim Preserve A(10, 10, 15)

If the size of the array is decreased, the data in deleted elements will be lost.

For other variables (not arrays) redeclaring means modyfing the type of the variable, while the original value is preserved or rounded.

When redeclaring variables-objects not equal to **Nothing**, the object is destroyed if it was created using the **New** statement, and the variable was the last link to it. The variable is also reset to **Nothing**.

Example

This example uses the ReDim statement to allocate and reallocate storage space for dynamic-array variables. It also shows how a variable can be redeclared to a new type.

```
Dim MyArray() As Integer ' Declare dynamic array.
Redim MyArray(5)     ' Allocate 5 elements.
For I = 1 To 5       ' Loop 5 times.
    MyArray(I) = I   ' Initialize array.
Next I
'The next statement resizes the array and erases the elements.
Redim MyArray(10)    ' Resize to 10 elements.
For I = 1 To 10      ' Loop 10 times.
    MyArray(I) = I   ' Initialize array.
Next I
'The following statement resizes the array but does not erase elements.
Redim Preserve MyArray(15) ' Resize to 15 elements.
'The next statement declares variable A as Integer
Dim A As Integer
A = 10              'Initialize A
Trace "A= " & A     ' trace A
```

```
ReDim A As Double    'Redeclare A as Double
Trace "A= " & A      ' trace A
```

See Also Data Type Summary, Dim Statement, Set Statement, Static Statement, Const Statement

Reset Statement

Reset Statement

Closes all files opened with the **Open** statement.

Syntax

Reset

Remarks

The **Reset** statement closes all active files that have been opened with **Open**, and writes all file buffers on the disk.

Example

In this example the Reset statement is used to close all open files and write their buffers on the disk. Note that the FileNumber variable of the Variant type is used as string and number at the same time.

```
Dim FileNumber
For FileNumber = 1 To 5     ' Repeat loop 5 times.
    ' Open file for writing. FileNumber is added
    ' to the filename as string and at the same time
    ' serves as the loop counter.
    Open "TEST" & FileNumber For Output As #FileNumber
    Write #FileNumber, "Hello All" ' Writing data into the file.
Next FileNumber
Reset                       ' Close all files and write the contents of the
buffers
                            'on the disk.
```

See Also **Close Statement**, **End Statement** , **Open Statement**

Resume Statement

Resume Statement

Resumes execution after an error-handling routine is finished.

Syntax

Resume

Resume Next

Resume *line*

The **Resume** statement syntax can have any of the following forms:

Statement	Description
Resume	If the error occurred in the same procedure as the error handler, execution resumes with the statement that caused the error. If the error occurred in a called procedure, execution resumes at the statement that last called out of the procedure containing the error-handling routine.
Resume Next	If the error occurred in the same procedure as the error handler, execution resumes with the statement immediately following the statement that caused the error. If the error occurred in a called procedure, execution resumes with the statement immediately following the statement that last called out of the procedure containing the error-handling routine (or **On Error Resume Next** statement).
Resume *line*	Execution resumes at *line* specified in the required *line* argument. The line argument is a line label or line number and must be in the same procedure as the error handler.

Remarks

If you use a **Resume** statement anywhere except in an error-handling routine, an error occurs.

Example

This example uses the **Resume** statement to end error handling in a procedure, and then resume execution with the statement that caused the error. Error number 75 is generated to illustrate using the **Resume** statement.

```
Sub ResumeStatementDemo()
   On Error GoTo ErrorHandler   ' Enable error-handling routine.
   Open "TESTFILE" For Output As #1   ' Open file for output.
   Kill "TESTFILE"   ' Attempt to delete open file.
   Exit Sub   ' Exit Sub to avoid error handler.
ErrorHandler:   ' Error-handling routine.
   Select Case Err()   ' Evaluate error number.
      Case 55,75   ' "File already open" or "Path/File access error" error.
         Trace """File already open"" or ""Path/File access error"" error"
         Close #1   ' Close open file.
      Case Else
         ' Handle other situations here....
   End Select
   Resume   ' Resume execution at same line that caused the error.
End Sub
```

See Also Erl Function , Err Function , Error$ Function , On Error Statement

Right Function

Right Function

Returns a **FixStr** (**String**) containing a specified number of characters from the right side of a string.

Syntax

Right(*string*, *length*)

The **Right** function syntax has these named arguments:

Element	Description
string	Required. String expression from which the rightmost characters are returned. If *string* contains **Null**, **Null** is returned.
length	Required; **Long**. Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in *string*, the entire string is returned.

Remarks

To determine the number of characters in *string*, use the **Len** function.

The **Right$** form returns **String** values. The **Right** form returns **FixStr** values.

Example

This example uses the **Right** function to return a specified number of characters from the right side of a string.

```
Dim AnyString, MyStr
AnyString = "Hello World"       ' Define string.
MyStr = Right(AnyString, 1)     ' Returns "d"
MyStr = Right(AnyString, 6)     ' Returns "World".
MyStr = Right(AnyString, 20)    ' Returns "Hello World".
```

See Also **Len Function**, **Mid Function** , **Left Function**

RmDir Statement

RmDir Statement

Removes an existing directory or folder.

Syntax

RmDir *path*

Remarks

The required argument *path* is a string, defining the directory or folder to be removed. It may contain a drive name. If the drive is not specified, the **RmDir** statement removes the directory or folder on the current drive.

An attempt to remove with **RmDir** a directory or folder which contain files will generate an error. To delete all files from directory or folder you should use the **Kill** statement.

Example

In this example the RmDir statement is used to remove an existing directory or folder.

```
' Assume that MYDIR is an empty directory or folder.
RmDir "MYDIR"     ' Removes MYDIR.
```

See Also [ChDir Statement](), [MkDir Statement](), [Kill Statement](), [CurDir Function]()

Rnd Function

Rnd Function

Returns a **Single** containing a random number.

Syntax

Rnd[(*num*)]

The optional *num* argument is a **Single** or any valid numeric expression.

Retun Values

If *num* is	**Rnd() returns**
Less than zero	The same number every time, using *num* as the seed.
Greater than zero	The next random number in the sequence.
Equal to zero	The most recently generated number.
Not supplied	The next random number in the sequence.

Remarks

The **Rnd()** function returns a value less than 1 but greater than or equal to zero.

The value of *num* determines how **Rnd** generates a random number:

For any given initial seed, the same number sequence is generated because each successive call to the **Rnd** function uses the previous number as a seed for the next number in the sequence.

Before calling **Rnd**, use the **Randomize** statement without an argument to initialize the random-number generator with a seed based on the system timer.

To produce random integers in a given range, use this formula:

Int((*upperbound* - *lowerbound* + 1) * **Rnd()** + *lowerbound*)

Here, *upperbound* is the highest number in the range, and *lowerbound* is the lowest number in the range.

Note To repeat sequences of random numbers, call **Rnd** with a negative argument immediately before using **Randomize** with a numeric argument. Using **Randomize** with the same value for number does not repeat the previous sequence.

Example

This example uses the Rnd function to generate a random integer value from 1 to 6.

```
Dim MyValue
MyValue = Int((6 * Rnd()) + 1)    ' Generate random value between 1 and 6.
```

See Also **Randomize Statement**, **Timer Function**

Round Function

Round Function

Returns a value of the same type that is passed to it, rounded to a specified number of decimal places.

Syntax

Round([*num*,[*NumAfterDecimal*]])

The optional *num* argument is a **Double** or any valid numeric expression. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 0.

The optional *NumAfterDecimal* argument is an **Int** indicating how many places to the right of the decimal are included in the rounding. The default value is 0, that is integers are returned **Round** function (the decimal separator is not displayed then).

Remarks

Use this function to get numbers of needed accuracy. If the number of places to the right of the decimal is greater than *NumAfterDecimal*, the last non-zero digit is rounded by standard mathematical rules. If the number of places to the right of the decimal in *num* is less than *NumAfterDecimal*, extra zeros are not displayed.

Example

This example shows how **Round** is used in 3 cases: when the second argument is omitted, when the number of places to the right of the decimal is greater than the second argument, and vice versa.

```
Dim MyNumber
MyNumber = Round(99.8)    ' Returns 100
MyNumber = Round(25.125, 2)    ' Returns 25.13
MyNumber = Round(-45.753, 5)    ' Returns -45.753
```

See Also [Fix Function](), [Int Function](), [Type Conversion Functions]()

RSet Statement

RSet Statement

Justifies the string by the right edge of the string variable.

Syntax

RSet *stringvar* = *string*

The **RSet** statement syntax has these Elements:

Element	**Description**
stringvar	Required. The name of the string variable.
string	Required. The string expression to be justified by the right edge of the string variable.

Example

In this example the RSet statement is used to justify the string by the right edge of the string variable.

```
Dim MyString
MyString = "0123456789"    ' Initializes the string.
Rset MyString = "Right->"  ' MyString contains " Right->".
Trace "|"&MyString&"|"
```

See Also [LSet Statement](), [Data Types]()

RTrim Function

Rtrim Function

Returns a **FixStr (String)** containing a copy of a specified string without trailing spaces.

Syntax
RTrim(*string*)

Remarks

The required *string* argument is any valid string expression. If *string* contains **Null**, **Null** is returned.

The **RTrim$** form returns **String** values. The **RTrim** form returns **FixStr** values.

Example

This example uses the **LTrim** function to strip leading spaces and the **RTrim** function to strip trailing spaces from a string variable. It uses the **Trim** function alone to strip both types of spaces.

```
Dim MyString, TrimString
MyString = " <-Trim-> "              ' Initialize string.
TrimString = LTrim(MyString)         ' TrimString = "<-Trim-> ".
TrimString = RTrim(MyString)         ' TrimString = " <-Trim->".
TrimString = LTrim(RTrim(MyString))  ' TrimString = "<-Trim->".
' Using the Trim function alone achieves the same result.
TrimString = Trim(MyString)          ' TrimString = "<-Trim->".
```

See Also **Left Function** , **Right Function**

Seek Function

Seek Function

Returns a **Long** specifying the current read/write position within a file opened using the **Open** statement.

Syntax
Seek(*filenumber*)

The required *filenumber* argument is an **Integer** containing a validfile number.

Remarks

Seek returns a value between 1 and 2,147,483,647 (equivalent to 2^31 – 1), inclusive.

The following describes the return values for each file access mode.

Mode	Returned Value
Random	Number of the next record read or written
Binary, Output, Append, Input	Byte position at which the next operation takes place. The first byte in a file is at position 1, the second byte is at position 2, and so on.

Example

For files opened in other modes, **Seek** returns the byte position at which the next operation takes place. Assume TESTFILE is a file containing a few lines of text.

```
Dim MyChar
Open "TESTFILE" For Input As #1 ' Open file for reading.
Do While Not EOF(1)             ' Loop until end of file.
    Get(#1,MyChar)              ' Read next character of data.
    Trace Seek(1)
Loop
Close #1                        ' Close file.
```

See Also **Get Statement**, **Open Statement** , **Put Statment**, **Seek Statement**, **Loc Function**

Seek Statement

Seek Statement

Sets the position for the next read/write operation within a file opened using the **Open** statement.

Syntax

Seek [#]*filenumber*, *position*

The **Seek** statement syntax has the following Elements:

Element	Description
filenumber	Required. Any valid file number.

position	Required. Number in the range 1 – 2,147,483,647, that indicates where the next read/write operation should occur.

Remarks

Record numbers specified in **Get** and **Put** statements override file positioning performed by **Seek**. Performing a file-write operation after a **Seek** operation beyond the end of a file extends the file. If you attempt a **Seek** operation to a negative or zero position, an error occurs.

Example

In this example the Seek statement sets a new position in the file for the next read/write operation.

```
Dim MaxSize, NextChar, MyChar
Open "TESTFILE" For Input As #1 ' Opens file for reading.
MaxSize = LOF(1)                ' Determines file size in bytes.
' Subsequently reads all records starting from the last one.
For NextChar = MaxSize To 1 Step -1
    Seek #1, NextChar           ' Specifies the byte number.
    MyChar = Input(1, #1)       'Reads symbol.
Next NextChar
Close #1                        ' Closes file.
```

See Also	**Recording Data in a File**, **Get Statement**, **Open Statement**, **Put Statement**, **Loc Function**, **Seek Function**

Select Case Statement

Select Case Statement

Executes one of several groups of statements, depending on the value of an expression.

Syntax

Select Case *testexpression*
[**Case** *expressionlist-n*
[*statements-n*]] ...
[**Case Else**
[*elsestatements*]]

End Select

The **Select Case** statement syntax has these Elements:

Element	**Description**

testexpression	Required. Any expression.
expressionlist-n	Required if a **Case** appears. Delimited list of one or more of the following forms: *expression*, *expression* **To** *expression*, **Is** *comparisonoperator expression*. The **To** keyword specifies a range of values. If you use the **To** keyword, the smaller value must appear before **To**. Use the **Is** keyword with comparison operators (except **Is** and **Like**) to specify a range of values.
statements-n	Optional. One or more statements executed if *testexpression* matches any Element of *expressionlist-n*.
elsestatements	Optional. One or more statements executed if *testexpression* doesn't match any of the **Case** clause.

Remarks

If *testexpression* matches any **Case** *expressionlist* expression, the statements following that **Case** clause are executed up to the next **Case** clause, or, for the last clause, up to **End Select**. Control then passes to the statement following **End Select**. If *testexpression* matches an *expressionlist* expression in more than one **Case** clause, only the statements following the first match are executed.

The **Case Else** clause is used to indicate the *elsestatements* to be executed if no match is found between the *testexpression* and an *expressionlist* in any of the other **Case** selections. Although not required, it is a good idea to have a **Case Else** statement in your **Select Case** block to handle unforeseen *testexpression* values. If no **Case** *expressionlist* matches *testexpression* and there is no **Case Else** statement, execution continues at the statement following **End Select**.

You can use multiple expressions or ranges in each **Case** clause. For example, the following line is valid:

Case 1 To 4, 7 To 9, 11, 13, Is > MaxNumber

Note: The **Is** comparison operator is not the same as the **Is** keyword used in the **Select Case** statement.

You also can specify ranges and multiple expressions for character strings. In the following example, **Case** matches strings that are exactly equal to "everything", strings that fall between "nuts" and "soup" in alphabetic order, and the current value of TestItem:

Case "everything", "nuts" To "soup", TestItem

Select Case statements can be nested. Each nested **Select Case** statement must have a matching **End Select** statement.

Example

This example uses the **Select Case** statement to evaluate the value of a variable. The second **Case** clause contains the value of the variable being evaluated, and therefore only the statement associated with it is executed.

```
Dim Number
Number = 8   ' Initialize variable.
Select Case Number   ' Evaluate Number.
Case 1 To 5   ' Number between 1 and 5, inclusive.
   Trace "Between 1 and 5"
' The following is the only Case clause that evaluates to True.
Case 6, 7, 8   ' Number between 6 and 8.
   Trace "Between 6 and 8"
Case 9 To 10   ' Number is 9 or 10.
   Trace "Greater than 8"
Case Else   ' Other values.
   Trace "Not between 1 and 10"
End Select
```

See Also If...Then...Else Statement , On...GoTo Statement

SetAttr Statement

SetAttr Statement

Sets attribute information for a file.

Syntax

SetAttr *pathname*, *attributes*

The **SetAttr** statement syntax has these named arguments:

Element	**Description**
pathname	Required. String expression that specifies a file name - may include directory or folder, and drive.
attributes	Required. Constant or numeric expression, setting file attributes.

Values

Below are possible values of the *attributes* argument:

Constant	**Value**	**Description**
cdbNormal	0	Normal.

cdbReadOnly	1	Read Only.
cdbHidden	2	Hidden.
cdbSystem	4	System (only in Microsoft Windows)
cdbArchive	32	File was changed since last backup (only in Microsoft Windows)
cdbAlias	64	The filename is an alias (only on the Macintosh).

Notice: These constants are defined in the application. This means that their names can be used anywhere in your code in place of the actual values.

Example

In this example the SetAttr statement is used to set attribute information for a file.

```
' Sets the Hidden attribute.
SetAttr "TESTFILE", cdbHidden
' Sets Hidden and Read Only attributes.
SetAttr "TESTFILE", cdbHidden + cdbReadOnly
```

See Also **FileAttr Function**, **GetAttr Function**

Set Statement

Set Statement

Assigns an object reference to a variable or property.

Syntax

Set *objectvar* = {[**New**] *objectexpression* | **Nothing**}

The **Set** statement syntax has these Elements:

Element	Description
objectvar	Required. Name of the variable or property; follows standard variable naming conventions.
New	Optional. **New** is usually used during declaration to enable implicit object creation. When **New** is used with **Set**, it creates a new instance of the object. If *objectvar* contained a reference to an object, that reference is released when the new one is assigned. The **New** keyword can't be used to create new instances of any intrinsic data type and can't be used to create dependent objects.

objectexpression	Required. Expression consisting of the name of an object, another declared variable of the same object type, or a function or method that returns an object of the same object type.
Nothing	Optional. Discontinues association of *objectvar* with any specific object. Assigning **Nothing** to *objectvar* releases all the system and memory resources associated with the previously referenced object when no other variable refers to it.

Remarks

To be valid, *objectvar* must be an object type consistent with the object being assigned to it.

The **Dim**, **ReDim**, and **Static** statements only declare a variable that refers to an object. No actual object is referred to until you use the **Set** statement to assign a specific object.

The following example illustrates how **Dim** is used to declare an array with the type DRect. No instance of DRect actually exists. Set then assigns references to new instances of DRect to the myRects variable.

```
Dim myRects(4) As DRect
Set myRects(1) = New DRect
Set myRects(2) = New DRect
Set myRects(3) = New DRect
Set myRects(4) = New DRect
```

Generally, when you use **Set** to assign an object reference to a variable, no copy of the object is created for that variable. Instead, a reference to the object is created. More than one object variable can refer to the same object. Because such variables are references to the object rather than copies of the object, any change in the object is reflected in all variables that refer to it. However, when you use the **New** keyword in the **Set** statement, you are actually creating an instance of the object.

Example

This example uses the **Set** statement to assign object references to variables.

```
Dim YourShape As Shape, MyObject, MyStr
Set YourShape = thisDoc.ActivePage.DrawRect(100,100,600,400)
Set MyObject = YourObject       ' Assign object reference.
' MyObject and YourShape refer to the same object.
YourShape.Text = "Hello World"  ' Initialize property.
MyStr = MyObject.Text           ' Returns "Hello World".
' Discontinue association. MyObject no longer refers to YourShape.
Set MyObject = Nothing          ' Release the object.
```

See Also Dim Statement, ReDim Statement, Let Statement, Static Statement

Sgn Function

Sgn Function

Returns a **Variant** (**Integer**) indicating the sign of a number.

Syntax

Sgn([*num*])

The optional *num* argument is a **Double** or any valid numeric expression. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 0.

Return Values

If *num* is	**Sgn() returns**
Greater than zero	1
Equal to zero	0
Less than zero	-1

Remarks

The sign of the *num* argument determines the return value of the **Sgn** function.

Example

```
Dim MyVar1, MyVar2, MyVar3, MySign
MyVar1 = 12: MyVar2 = -2.4: MyVar3 = 0
MySign = Sgn(MyVar1)   ' Returns 1.
MySign = Sgn(MyVar2)   ' Returns -1.
MySign = Sgn(MyVar3)   ' Returns 0.
```

See Also **Abs Function**

Shell Function

Shell Function

Runs an executable program and returns a **Variant** (**Boolean**) if successful, representing the result.

Syntax

Shell([*pathname*[,*windowstyle*]])

The **Shell** function syntax has these named arguments:

Element	**Description**
pathname	Optional; **Variant** (**String**). Name of the program to execute and any required arguments or command-line switches; may include directory or folder and drive.
windowstyle	Optional. **Variant** (**Integer**) corresponding to the style of the window in which the program is to be run. If *windowstyle* is omitted, the program is started minimized with focus.

The *windowstyle* named argument has these values:

Constant	**Value**	**Description**
cdbHide	0	Window is hidden and focus is passed to the hidden window.
cdbNormalFocus	1	Window has focus and is restored to its original size and position.
cdbMinimizedFocus	2	Window is displayed as an icon with focus.
cdbMaximizedFocus	3	Window is maximized with focus.
cdbNormalNoFocus	4	Window is restored to its most recent size and position. The currently active window remains active.
cdbMinimizedNoFocus	6	Window is displayed as an icon. The currently active window remains active.

Remarks

If the **Shell** function successfully executes the named file, it returns **True**. If the **Shell** function can't start the named program, it returns **False**.

> **Note** The **Shell** function runs other programs asynchronously. This means that a program started with **Shell** might not finish executing before the statements following the **Shell** function are executed.
>
> The *windowstyle* argument is only considered on Windows system.

Example

```
' Specifying 1 as the second argument opens the application in
' normal size and gives it the focus.
Dim RetVal
RetVal = Shell("C:\WINDOWS\CALC.EXE", 1)    ' Run Calculator.
```

See Also [GetOpenFileName Function]

Sin Function

Sin Function

Returns a **Double** specifying the sine of an angle.

Syntax

Sin([*num*])

The optional *num* argument is a **Double** or any valid numeric expression, specifying the angle in radians. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 0.

Remarks

The **Sin** function takes an angle in radians and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse.

The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Example

```
Dim MyAngle, MyCosecant
MyAngle = 1.3   ' Define angle in radians.
MyCosecant = 1 / Sin(MyAngle)   ' Calculate cosecant.
```

See Also [Atn Function](), [Cos Function](), [Tan Function]()

Space Function

Space Function

Returns a **FixStr** (**String**) consisting of the specified number of spaces.

Syntax
Space[$]([*number*])

The required *number* argument is the number of spaces you want in the string.

Remarks

The **Space** function is useful for formatting output and clearing data in fixed-length strings. The **Space$** form **String** values. The **Space** form returns **FixStr** values.

Example

This example uses the **Space** function to return a string consisting of a specified number of spaces.

```
Dim MyString
' Returns a string with 10 spaces.
MyString = Space(10)
' Insert 10 spaces between two strings.
MyString = "Hi" & Space(10) & "all"
```

See Also Spc Function, String Function

Spc Function

Spc Function

Used with the **Print #** statement to position output.

Syntax

Spc(*n*)

The required *n* argument is the number of spaces to insert before displaying or printing the next expression in a list.

Remarks

If *n* is less than the output line width, the next print position immediately follows the number of spaces printed. If *n* is greater than the output line width, Spc calculates the next print position using the formula:

currentWritePosition + (n Mod width)

For example, if the current print position is 24, the output line width is 80, and you specify **Spc**(90), the next print will start at position 34 (current print position + the remainder of 90/80). If the difference between the current print position and the output line width is less than n (or *n* **Mod** *width*), the **Spc** function skips to the beginning of the next line and generates spaces equal to *n – (width – currentWritePosition)*.

Note. Make sure your tabular columns are wide enough to accommodate wide letters.

Example

In the example below the **Spc** function is used to position output in a file.

```
' The Spc function can be used with the Print # statement.
' Open file for output.
Open "TESTFILE" For Output As #1
Print #1, "10 space between this string"; Spc(10); "and this string."
Close #1    ' Close file.
```

See Also	**Print # Statement**, **Width # Statement**, **Mod Operator**, **Space Function** , **Tab Function**

Sqr Function

Sqr Function

Returns a **Double** specifying the square root of a number.

Syntax

Sqr([*num*])

The required *num* argument is a **Double** or any valid numeric expression greater than or equal to zero. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns **Null**.

Example
```
Dim MySqr
MySqr = Sqr(4)    ' Returns 2.
MySqr = Sqr(23)    ' Returns 4.79583152331272.
MySqr = Sqr(0)    ' Returns 0.
MySqr = Sqr(-4)    ' Generates a run-time error.
```

Static Statement

Static Statement

Declare static variables and allocate storage space. Variables declared with the **Static** statement retain their values as long as the code is running.

Syntax

Static *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*] [**,** *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*]] . . .

The **Static** statement syntax has these Elements:

Element	**Description**
varname	Required. Name of the variable; follows standard variable naming conventions.
subscripts	Optional. Dimensions of an array variable; up to 10 multiple dimensions may be declared. The subscripts argument uses the following syntax: *count1*[, *count2*] . . . where *count1*, *count2* are constants, indicating the upper limit of allowable indices for the defined array. The lower limit of allowable indices always equals 0. So, for a one-dimensional array the number of elements can be calculated as *count1*+1 .
New	Optional. Keyword that enables implicit creation of an object. If you use **New** when declaring the object variable, a new instance of the object is created during declaration, so you don't have to use the **Set** statement to assign the object reference. The **New** keyword can't be used to declare variables of any intrinsic data type, can't be used to declare instances of dependent objects or objects that don't have built-in constructor.
type	Optional. Data type of the variable; may be Byte, Boolean, Integer, Long, Single, Double, Date, String (for variable-length strings), **String** * *length* (for fixed-length strings), Object, Variant, or an object type. Use a separate **As** *type* clause for each variable you declare.

Remarks

Once module code is running, variables declared with the **Static** statement retain their value until the module is reset or restarted. Use the **Static** statement in procedures to explicitly declare variables that are visible only within the procedure, but whose lifetime is the same as the module in which the procedure is defined.

Use a **Static** statement within a procedure to declare the data type of a variable that retains its value between procedure calls. For example, the following statement declares a fixed-size array of integers:

Static EmployeeNumber(200) As Integer

The following statement declares a variable for a new instance of a database engine:

Static Eng As New dbEngine

If the **New** keyword isn't used when declaring an object variable, the variable that refers to the object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**, which indicates that it doesn't refer to any Elementicular instance of an object. When you use the **New** keyword in the declaration, an instance of the object will be created.

If you don't specify a data type or object type, the variable is Variant by default.

All declared variables except objects declared with **New**, take the **Empty** value, which indicates that they are not initialized.

Tip: It's recommended to place all delcarations in the beginning of a module or a procedure. This shortens the time of compilation.

Example

This example uses the **Static** statement to retain the value of a variable for as long as module code is running.

```
' Function definition.
Function KeepTotal(Number As Long) As Long
    ' Variable Accumulate preserves its value between calls.
    Static Accumulate As Long
    Accumulate = Accumulate + Number
    KeepTotal = Accumulate
End Function
```

See Also Data Type Summary, Dim Statement, ReDim Statement, Set Statement, Const Statement, Function Statement, Sub Statement

Statements Index

- Let
- Line Input
- LSet

- Mid
- MkDir

- Name

- On...GoSub
- On...GoTo
- On Error
- Open

- Pause
- Print #
- Put

- Randomize
- ReDim
- Reset
- Resume
- RmDir
- RSet

- Seek
- Select Case
- Set
- SetAttr
- Static
- Stop
- Sub...End Sub

- Time=
- Trace

- Wait

- While...Wend
- Width #

- Write #

Statements

Statements

Category	Actions	Statements
Arrays	Declare and initialize	Dim, ReDim, Static
	Reinitialize	Erase, ReDim
Controlling program flow	Branch	GoSub...Return, GoTo, On Error, On...GoSub, On...GoTo
	Exit or pause the program	End, Stop, Wait, Pause
	Loop	Do...Loop, For...Next, While...Wend
	Make decision	If...Then...Else, Select Case
	Preprocessor directives	#If...#Else...#Endif
Date/time	Set date or time	Date=, Time=
Error trapping	Simulate run-time errors	Error
	Trap errors while a program is running	On Error, Resume
File I/O	Access or create a file	Open
	Close files	Close, Reset
	Control output appearance	Width #
	Copy one file to another	FileCopy
	Manage disk drives or directories	ChDir, ChDrive, MkDir, RmDir
	Manage files	Kill, Name
	Read from file	Get, Input, Line Input
	Set file attributes	SetAttr
	Set read-write position in a file	Seek
	Write to a file	Print, Put, Write #
Math	Generate random numbers	Randomize
Procedures	Call a procedure	Call
	Declare a reference to an external procedure	Declare
	Define a procedure	Function...End Function, Sub...End Sub
	Exit from a procedure	Exit Function, Exit Sub
Strings	Justify a string	LSet, RSet
	Manipulate strings	Mid

Variables and constants	Declare variables or constants	Const, Enum, Dim, Static
	Assign value	Let, Set
Miscellaneous	Sound a beep	Beep
	Tracing	Trace
Index	Alphabetical list of statements	

Stop Statement

Stop Statement

Suspends execution.

Syntax
Stop

Remarks

You can place **Stop** statements anywhere in the code to suspend execution.

The **Stop** statement suspends execution, but unlike **End**, it doesn't close any files or clear variables. Also unlike **End**, it doesn't stop execution of lower-level scripts.

If it's necessary to suspend execution of the script, but leave its procedures waiting for subsequent calls, you should use the **Stop** statement.

Note: The **Stop** statement operates within one execution level of ConceptDraw Basic script. For example, you can define at the document level your procedures for common use from different execution levels, and suspend execution of the code at the document level with **Stop**. Then you can use procedures of the document level in code at the page or shape level.

The code at the execution level where **Stop** was performed is suspended and remains resident waiting until procedures defined in it are called.

Note: If neither **Stop** nor **End** was met during execition of code, code of this execution level is considered resident by default.

Example

The following example demonstrates how to leave resident procedures of any execution level in ConceptDraw Basic. Here the gData array is defined in the global area, and three procedures are defined in the code. However initially only one procedure - InitGlobalData() is called from the global area. The execution is suspened by the **Stop** statement, leaving all procedures resident waiting for subsequent calls.

```
Dim gData(256) As Double
' Definition of InitGlobalData() procedure
Sub InitGlobalData()
        ' Make global data initialization
        For i = 0 To 256
              gData(i)=i
        Next
End Sub
' Definition of TraceGlobalData() procedure
Sub TraceGlobalData ()
        For i = 0 To 256
        Trace gData(i)
        Next
End Sub
' Definition of RecalcGlobalData() procedure
Sub RecalcGlobalData ()
        For i = 0 To 256
              ' Do some calculation here
              gData(i)=gData(i)+Rnd()
        Next
End Sub
InitGlobalData() ' Call procedure for global data initialization
Stop
```

See Also End Statement

StrComp Function

StrComp Function

Returns an **Integer** indicating the result of a string comparison.

Syntax

StrComp(*string1*, *string2*[, *compare*])

Remarks

The **StrComp** function syntax has these named arguments:

Element	Description

string1	Required. Any valid string expression.
string2	Required. Any valid string expression.
compare	Optional. Specifies the type of string comparison. The *compare* argument may be omitted or have 0 or 1 value. To perform binary comparison, specify 0 (default). To perform not case-sensitive textual comparison, specify 1.

Return Values

If	**StrComp returns**
string1 is less than *string2*	-1
string1 is equal to *string2*	0
string1 is greater than *string2*	1
string1 or *string2* is **Null**	**Null**

Example

This example uses the **StrComp** function to return the results of a string comparison. If the third argument is 1, a textual comparison is performed; if the third argument is 0 or omitted, a binary comparison is performed.

```
Dim MyStr1, MyStr2, MyComp
MyStr1 = "ABCD": MyStr2 = "abcd"        ' Define variables.
MyComp = StrComp(MyStr1, MyStr2, 1)     ' Returns 0.
MyComp = StrComp(MyStr1, MyStr2, 0)     ' Returns -1.
MyComp = StrComp(MyStr2, MyStr1)        ' Returns 1.
```

See Also InStr Function

String Function

String Function

Returns a **FixStr** (**String**) containing a repeating character string of the length specified.

Syntax

String(*number*, *character*)

The **String** function syntax has these named arguments:

Element	Description
number	Required; **Long**. Length of the returned string. If *number* contains **Null**, **Null** is returned.
character	Required; **Variant**. Character code specifying the character or string expression whose first character is used to build the return string. If *character* contains **Null**, **Null** is returned.

Remarks

If you specify a number for *character* greater than 255, **String** converts the number to a valid character code using the formula:

character **Mod** 256

The **String$** form returns **String** values. The **String** form returns **FixStr** values.

Example

This example uses the **String** function to return repeating character strings of the length specified.

```
Dim MyString
MyString = String(5, "*")       ' Returns "*****"
MyString = String(5, 42)        ' Returns "*****"
MyString = String(10, "ABC")    ' Returns "AAAAAAAAAA"
```

See Also **Mod Operator**, **Space Function**

Str Function

Str Function

Return an **FixStr** value.

Syntax

Str[$]([*number*])

Example

```
Dim MyString
```

```
MyString = Str(459)    ' Returns " 459".
MyString = Str(-459.65)    ' Returns "-459.65".
MyString = Str(459.001)    ' Returns " 459.001".
```

See Also Format Function, Type Conversion Functions, Val Function

Sub...End Sub Statement

Sub...End Sub Statement

Declares the name, arguments, and code that form the body of a **Sub** procedure.

Syntax

Sub *name* ([*arglist*])
[*statements*]
[**Exit Sub**]
[*statements*]

End Sub

The **Sub** statement syntax has these Elements:

Element	Description
name	Required. Name of the **Sub**; follows standard variable naming conventions.
arglist	Optional. List of variables representing arguments that are passed to the **Sub** procedure when it is called. Multiple variables are separated by commas.
statements	Optional. Any group of statements to be executed within the **Sub** procedure.

The *arglist* argument has the following syntax and Elements:
[**ByVal** | **ByRef**] *varname* [**As** *type*] [=*defval*]

Element	Description
ByVal	Optional. Indicates that the argument is passed by value. **ByVal** is the default in ConceptDraw Basic.
ByRef	Optional. Indicates that the argument is passed by reference.

varname	Required. Name of the variable representing the argument being passed to the procedure; follows standard variable naming conventions.
type	Optional. Data type of the valuepassed to the procedure; may be Byte, Boolean, Integer, Long, Single, Double, Date, String (except fixed length), Object , Variant or an object type.
defval	Optional. Constant that determine the value that will be passed to the procedure by default if this argument is omitted.

Remarks

Sub procedures can be recursive; that is, they can call themselves to perform a given task. However, recursion can lead to stack overflow.

The **Exit Sub** keywords cause an immediate exit from a **Sub** procedure. Program execution continues with the statement following the statement that called the **Sub** procedure. Any number of **Exit Sub** statements can appear anywhere in a **Sub** procedure.

Like a **Function** procedure, a **Sub** procedure is a separate procedure that can take arguments, perform a series of statements, and change the value of its arguments. However, unlike a **Function** procedure, which returns a value, a **Sub** procedure can't be used in an expression.

You call a **Sub** procedure using the procedure name followed by the argument list. See the **Call** statement for specific information on how to call **Sub** procedures.

Variables used in **Sub** procedures fall into two categories: those that are explicitly declared within the procedure and those that are not. Variables that are explicitly declared in a procedure (using **Dim** or the equivalent) are always local to the procedure. Variables that are used but not explicitly declared in a procedure are also local unless they are explicitly declared at some higher level outside the procedure.

Note: You can't use **GoSub**, **GoTo**, or **Return** to enter or exit a **Sub** procedure.

Example

This example uses the **Sub** statement to define the name, arguments, and code that form the body of a **Sub** procedure.

```
' Sub procedure definition.
' Sub procedure with two arguments.
Sub SubTraceXY(x As Double, y As Double)
   Trace "X = " & x & "  Y= " & y   ' Print x,y to Output window.
End Sub
```

See Also Call Statement , Dim Statement , Exit Statement , Function Statement

Tab Function

Tab Function

Used with the **Print #** statement to position output.

Syntax

Tab[(*n*)]

The optional *n* argument is the column number moved to before displaying or printing the next expression in a list. If omitted, **Tab** moves the insertion point to the beginning of the next print zone. This allows **Tab** to be used instead of a comma in locales where the comma is used as a decimal separator.

Remarks

If the current print position on the current line is greater than *n*, **Tab** skips to the *n*th column on the next output line. If *n* is less than 1, **Tab** moves the print position to column 1. If *n* is greater than the output line width, **Tab** calculates the next print position using the formula:

n **Mod** width

For example, if width is 80 and you specify **Tab**(90), the next print will start at column 10 (the remainder of 90/80). If *n* is less than the current print position, printing begins on the next line at the calculated print position. If the calculated print position is greater than the current print position, printing begins at the calculated print position on the same line.

The leftmost print position on an output line is always 1. When you use the **Print #** statement to print to files, the rightmost print position is the current width of the output file, which you can set using the **Width #** statement.

Note. Make sure your tabular columns are wide enough to accommodate wide letters.

Example

This example uses the **Tab** function to position output in a file.

```
' The Tab function can be used with the Print # statement.
Open "TESTFILE" For Output As #1 ' Open file for output.
' The second word prints at column 20.
Print #1, "Hello"; Tab(20); "World!"
' If the argument is omitted, cursor is moved to the next print zone.
Print #1, "Hello"; Tab(1); "World!"
Close #1 ' Close file.
```

See Also Print # Statement, Width # Statement, Mod Operator, Space Function , Spc Function

Tan Function

Tan Function

Returns a **Double** specifying the tangent of an angle.

Syntax

Tan([*num*])

The optional *num* argument is a **Double** or any valid numeric expression, specifying the angle in radians. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 0.

Remarks

Tan takes an angle in radians and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Example

```
Dim MyAngle, MyCotangent
MyAngle = 1.3    ' Define angle in radians.
MyCotangent = 1 / Tan(MyAngle)    ' Calculate cotangent.
```

See Also Atn Function, Cos Function, Sin Function

Timer Function

Timer Function

Returns a **Double** representing the number of seconds elapsed since system was started.

Syntax
Timer()

Example
```
Dim Start, Finish, Res
Start = Timer()
For a = 1 to 1000000
Res = sqr(a)
Next a
Finish = Timer()
MsgBox(Finish-Start)    ' Loop run time in seconds
```

See Also Randomize Statement, Time Function

Time Function

Time Function

Returns a **Date** (**String**) indicating the current system time.

Syntax
Time[$]()

Remarks

The **Time$** form returns **String** values. The **Time** form returns **Date** values. Use the **Time** statement to set system date.

Example
```
Dim MyTime
MyTime = Time()     ' Assign current system time
```

See Also Date Function, Date Statement, Time Statement, Timer Function

Time= Statement

Time= Statement

Sets the system time.

Syntax

Time = *time*

Remarks

If *time* is a string, **Time** attempts to convert it to a time using the time separators you specified for your system. If it can't be converted to a valid time, an error occurs.

Note: Changing time is only possible if you have enough rights, required by the system.

Example

```
Dim MyTime
MyTime = #4:35:17 PM#    ' Assign a time.
Time = MyTime    ' Set system time to MyTime.
```

See Also Date Function, Date Statement, Time Function

Trace Statement

Trace Statement

Outputs information in the Output Window

Syntax

Trace *expression*

Remarks

Outputs the value of *expression* to "CDBasic Output" window. Trace statement is used to trace the values of expressions and variables during debugging.

Example

```
Dim str as string
```

```
str = "test message"
TRACE str
```

See Also MsgBox Function

Trim Function

Trim Function

Returns a **FixStr (String)** containing a copy of a specified string without leading and trailing spaces.

Syntax
Trim(*string*)

Remarks

The required *string* argument is any valid string expression. If *string* contains **Null**, **Null** is returned.

The **Trim$** form returns **String** values. The **Trim** form returns **FixStr** values.

Example

This example uses the **LTrim** function to strip leading spaces and the **RTrim** function to strip trailing spaces from a string variable. It uses the **Trim** function to strip both types of spaces.

```
Dim MyString, TrimString
MyString = " <-Trim-> "              ' Initialize string.
TrimString = LTrim(MyString)         ' TrimString = "<-Trim-> ".
TrimString = RTrim(MyString)         ' TrimString = " <-Trim->".
TrimString = LTrim(RTrim(MyString))  ' TrimString = "<-Trim->".
' Using the Trim function alone achieves the same result.
TrimString = Trim(MyString)          ' TrimString = "<-Trim->".
```

See Also **Left Function**, **Right Function**

UCase Function

UCase Function

Returns a **FixStr** (**String**) that has been converted to uppercase.

Syntax
UCase[$](*string*)

The required *string* argument is any valid string expression. If string contains **Null**, **Null** is returned.

Remarks

Only lowercase letters are converted to uppercase; all uppercase letters and nonletter characters remain unchanged.

The **UCase$** form returns **String** values. The **UCase** form returns **FixStr** values.

Example

This example uses the **UCase** function to return an uppercase version of a string.
```
Dim LowerCase, UpperCase
LowerCase = "Hi all 1234"  ' String to convert.
UpperCase = UCase(LowerCase)    ' Returns "HI ALL 1234".
```

See Also [LCase Function]

Val Function

Val Function

Returns the numbers contained in a string as a numeric value of appropriate type.

Syntax
Val([*string*])

The optional *string* argument is any valid string expression. If this argument is omitted, is a non-initialized variable, or **Null**, the function returns 0.

Remarks

The **Val** function stops reading the *string* at the first character it can't recognize as Element of a number. Symbols and characters that are often considered Elements of numeric values, such as dollar signs and commas, are not recognized. However, the function recognizes the radix prefixes &O (for octal) and &H (for hexadecimal). Blanks, tabs, and linefeed characters are stripped from the argument.

The following returns the value 1835:

Val(" 1 835 dollars 28 cents")

In the code below, **Val** returns the decimal value -1 for the hexadecimal value shown:

Val("&HFFFF")

Use the **IsDate** function to determine if *date* can be converted to a date or time. **CVDate** recognizes date literals and time literals as well as some numbers that fall within the range of acceptable dates. When converting a number to a date, the whole number portion is converted to a date. Any fractional Element of the number is converted to a time of day, starting at midnight.

> **Note** The **Val** function recognizes only the period (.) as a valid decimal separator. When different decimal separators are used, as in international applications, use **CDbl** instead to convert a string to a number.

Example

```
Dim MyValue
MyValue = Val("2457")    ' Returns 2457.
MyValue = Val(" 2 45 7")    ' Returns 2457.
MyValue = Val("24 and 57")    ' Returns 24.
```

See Also **Str Function**, **Type Conversion Functions**

VarType Function

VarType Function

Returns an Integer indicating the type or subtype of a variable.

Syntax

VarType(*varname*)

The required *varname* argument is any variable.

Return Values

Constant	Value	Description
cdbEmpty	0	Empty (uninitialized). Returns for Variant only.
cdbNull	1	Null
cdbInteger	2	Integer
cdbLong	3	Long integer
cdbSingle	4	Single-precision floating-point number
cdbDouble	5	Double-precision floating-point number
cdbDate	7	Date value
cdbString	8	String
cdbObject	9	Object
cdbBoolean	11	Boolean value
cdbByte	17	Byte value

Note: These constants are specified by ConceptDraw Basic. The names can be used anywhere in your code in place of the actual values.

Example

```
Dim IntVar, StrVar, DateVar, MyCheck
' Initialize variables.
IntVar = CInt(459)
StrVar = "Hello World"
DateVar = #2/12/69#
MyCheck = VarType(IntVar)    ' Returns 2.
Trace MyCheck
MyCheck = VarType(DateVar)    ' Returns 7.
Trace MyCheck
MyCheck = VarType(StrVar)    ' Returns 8.
Trace MyCheck
```

See Also Data Type Summary , IsDate Function , IsEmpty Function , IsNull Function , IsNumeric Function

Wait Statement

Wait Statement

Suspends the execution of the script for a specified interval.

Syntax

Wait *TimeoutMilliseconds*

The *TimeoutMilliseconds* parameter specifies the time, in milliseconds, for which to suspend execution.

Remarks

The **Wait** statement is fully equivalent to the **Pause** statement. These two statements work absolutely identically and are supported for compatibility with different versions of BASIC.

Example

In the example below **Wait** is used to create a 5 second pause.

```
Wait 5000
```

See Also Pause Statement, Timer Function

While...Wend Statement

While...Wend Statement

Executes a series of statements as long as a given condition is **True**.

Syntax

While *condition*
[*statements*]

Wend

The **While...Wend** statement syntax has these Elements:

Element	Description
condition	Required. Expression that is **True** or **False**.
statements	One or more statements that are repeated while, or until, *condition* is **True**.

Remarks

If *condition* is **True**, all statements are executed until the **Wend** statement is encountered. Control then returns to the **While** statement and *condition* is again checked. If *condition* is still **True**, the process is repeated. If it is not **True**, execution resumes with the statement following the **Wend** statement.

While...Wend loops can be nested to any level. Each **Wend** matches the most recent **While**.

Example

This example uses the **While...Wend** statement to increment a counter variable. The statements in the loop are executed as long as the condition evaluates to **True**.

```
Dim Counter
Counter = 0   ' Initialize variable.
While Counter < 20   ' Test value of Counter.
   Counter = Counter + 1   ' Increment Counter.
Wend   ' End While loop when Counter > 19.
Trace Counter   ' Prints 20 in the Output window.
```

See Also Do..Loop Statement, For...Next Statement

Width # Statement

Width # Statement

Set the string width for the files opened with the **Open** statement.

Syntax

Width *#filenumber, width*

The **Width** # sintax contains these Elements:

Element	Description

filenumber	Required. Any valid file number.
width	Required. Numeric expression with the 0–255 range inclusive. Sets how many symbols to write to a line before going over to a new line. If *width* equals 0 (default value), line width is not limited. The default value of the *width* argument is 0.

Example

In the example below the Width # statement is used to set the line width for writing to the file.

```
Dim I
Open "TESTFILE" For Output As #1   ' Open file for writing.
Width #1, 5                        ' Set width to  5.
For I = 0 To 9                     ' The loop repeats 10 times.
    Print #1, Chr(48 + I);         ' Print 5 characters per line.
Next I
Close #1                           ' Close the file.
```

See Also **Open Statement**, **Print # Statement**

Write # Statement

Write # Statement

Writes non-formatted data to a sequential file.

Syntax

Write #*filenumber*, [*outputlist*]

The **Write** # statement syntax has the following Elements:

Element	Description
filenumber	Required. Any valid file number.
outputlist	Optional. One or more comma-delimited numeric expressions or string expressions to write to a file.

Remarks

Data written with **Write #** is usually read from a file with **Input #**.

If you omit *outputlist* and include a comma after *filenumber*, a blank line is printed to the file. Multiple expressions can be separated with a space, a semicolon, or a comma. A space has the same effect as a semicolon.
When **Write #** is used to write data to a file, several universal assumptions are followed so the data can always be read and correctly interpreted using **Input #**, regardless of local settings:

- Numeric data is always written using the period as the decimal separator.
- For **Boolean** data, either #TRUE# or #FALSE# is printed. The **True** and **False** keywords are not translated.
- Date data is written to the file using the universal date format. When either the date or the time component is missing or zero, only the Element provided gets written to the file.
- If *outputlist* is **Null** or **Empty**, #NULL# is written to the file.

Unlike the **Print #** statement, the **Write #** statement inserts commas between items and quotation marks around strings as they are written to the file. You don't have to put explicit delimiters in the list. **Write #** inserts a newline character, that is, a carriage return–linefeed (Chr(13) + Chr(10)), after it has written the final character in *outputlist* to the file.

Example

In this example the the Write # statement is used to write non-formatted data to a sequential file.

```
Open "TESTFILE" For Output As #1   ' Opens file for writing.
Write #1, "Hello World", 234       ' Writes comma-delimited data.
Write #1,                          ' Writes a blank line.
Dim MyBool, MyDate, MyNull
' Assigns values of Boolean, Date, Null types.
MyBool = False
MyDate = #February 12, 1969#
MyNull = Null
' Boolean data gets written as #TRUE# or #FALSE#. Date data is written
' using the universal date format, for instance,  #1994-07-13#
' for July 13, 1994. Null values are written as #NULL#.
Write #1, MyBool ; " - logical"
Write #1, MyDate ; " - Date"
Write #1, MyNull ; " - Null"
Close #1 ' Closes file.
```

See Also **Writing Data to a File**, **Input # Statement**, **Open Statement**, **Print # Statement**

XOR Operator

XOR Operator

Used to perform a logical exclusion on two expressions.

Syntax

[*result* =] *expression1* Xor *expression2*

The Xor operator syntax has these Elements:

Element	**Description**
result	Optional; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If one, and only one, of the expressions evaluates to **True**, result is **True**. The following table illustrates how result is determined:

If expression1 is	**And expression2 is**	**The result is**
True	True	False
True	False	True
False	True	True
False	False	False

The Xor operator performs as both a logical and bitwise operator. A bit-wise comparison of two expressions using exclusive-or logic to form the result, as shown in the following table:

If expression1 is	**And expression2 is**	**The result is**
0	0	0
0	1	1
1	0	1
1	1	0

Example

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null   ' Initialize variables.
MyCheck = A > B Xor B > C   ' Returns False.
trace MyCheck
MyCheck = B > A Xor B > C   ' Returns True.
trace MyCheck
MyCheck = B > A Xor C > B   ' Returns False.
trace MyCheck
MyCheck = B > D Xor A > B   ' Returns False.
trace MyCheck
MyCheck = A Xor B   ' Returns 2 (bitwise comparison).
trace MyCheck
```

See Also **Operators**

Objects Reference

- ConceptDraw access Objects
- Database access Objects

ConceptDraw access Objects

Action Object

Action Object

The **Action** object is used to control the properties of a ConceptDraw shape's user-defined menu item and the action, associated with it. The user-defined menu appears when you right-click on the shape. You can add items to the menu and assign to an item a formula which will be executed when this menu item is clicked. To assign a formula to a menu item, use the **SetPropertyFormula** method of the **Shape** object. The following methods of the **Shape** object are defined for working with the menu item collection of a shape's user-defined menu:

Properties

Action	The result of executing the formula, associated with this menu item.
Menu	The name of the menu item.
Prompt	The prompt that appears in the status bar when the menu item is selected.
Checked	The state of a check mark beside the command name on the menu
Disabled	The state of a menu item

Remarks

To retrieve an instance of the **Action** object, corresponding to an item of the user-defined menu item collection of the shape, use the **Action** method of the **Shape** object. Use the **ActionsNum** method to find out the number of the user-defined menu items. The **AddAction** method can be used to add an item to the user-defined menu, and **RemoveAction** - to remove one.

See Also Action method, ActionsNum method, AddAction method, RemoveAction method, Shape object

Application Object

The **Application** object is used to control and get information about the ConceptDraw application. By using the methods and properties of this object you can create new documents and libraries, open, close and save the existing ones, control the user-defined menu at the application level, library windows of the application and many more.

Properties

ActiveDoc	Read-only. Returns the active document of the application.
ActiveLib	Read-only. Returns the active library.
ActiveLibWnd	Read-only. Returns the active library window.
CustomMenu	Read-only. Returns the user-defined menu of the application.
DocumentsPath	Read-only. Returns the full path to the files that are on the way, in a dialogue nastraevaemom **Preferences** application in the **Paths** tab in the **Documents.**
HelpPath	Read-only. Returns the full path to the files that are on the way, in a dialogue nastraevaemom **Preferences** application in the **Paths** tab in the **Help.**
LibrariesPath	Read-only. Returns the full path to the files that are on the way, in a dialogue nastraevaemom **Preferences** application in the **Paths** tab in the **Libraries.**
TemplatesPath	Read-only. Returns the full path to the files that are on the way, in a dialogue nastraevaemom **Preferences** application in the **Paths** tab in the **Templates.**

Methods

CloseDoc	Closes a document.
CloseLib	Closes a library.
CreateNewDoc	Creates a new document.
CreateNewLib	Creates a new library.
Doc	Returns a document by its index in the document collection of the application.
DocByName	Searches a document by its name (the **Name** property) in the document collection of the application.
DocsNum	Returns the number of open documents.
FindLib	Returns the index of the library in the library collection of the document.
FirstDoc	Returns the first document in the document collection of the application.
FirstLibWindow	Returns the first library window in the library window collection of the application.
Import	Imports a file of any format supported in ConceptDraw

Lib	Returns a library by its index in the library collection of the application.
LibByName	Searches a library by the specified name (the **Name** property) in the library collection of the application.
LibsNum	Returns the number of open libraries.
LibWindowByID	Returns the library window by its ID.
LibWindowsNum	Returns the number of library windows in the application.
NextDoc	Returns the next document in the document collection of the application.
NextLibWindow	Returns the next library window in the library window collection of the application.
OpenDoc	Opens an existing ConceptDraw document file.
OpenLib	Opens an existing ConceptDraw library file.
OpenWorkspace	Opens an existing ConceptDraw workspace file.
SaveWorkspace	Saves the current workspace in a file.
SetActiveLib	Makes the specified library the active library.

Remarks

An instance of the **Application** object can be retrieved by using the **thisApp** global variable, which returns an instance of the application in which the script is being executed. This variable is accessible at all ConceptDraw Basic script levels.

See Also Document object, Library object, Menu object, Window object

ConceptDraw access Objects Methods Index

ConceptDraw access Objects Methods Index

- Action
- ActionsNum
- AddAction
- AddConnectDot
- AddControlDot
- AddCustomProp
- AddDataSource
- AddDSValue
- AddGeometry
- AddHyperlinkToDocument
- AddHyperlinkToFile
- AddHyperlinkToPageShape
- AddHyperlinkToURL
- AddLayer
- AddMaster
- AddMenuItem
- AddPage
- AddStyle
- AddTabStop
- AddVariable
- ArcTo

- BeginShape

- Character
- CharactersNum
- CloseDoc
- CloseLib
- ColCount
- ColorEntry
- ColorProperty
- ColorsNum
- ConnectDot
- ConnectDotsNum
- ControlDot
- ControlDotsNum
- ConvertToGroup
- ConvertToVFPicture
- CreateNewDoc
- CreateNewLib
- CSVColorValue
- CSVGetColumnForKey
- CSVMinRowLength
- CSVRowLength

ConceptDraw access Objects Properties Index

ConceptDraw access Objects Properties Index

ConceptDraw access Objects

ConceptDraw access Objects

- Properties
- Methods
- Constants

Object Name	**Description**
Action	Provides access to the user-defined menu items. An instance of the **Action** object can be retrieved by using the methods of the **Shape** object.
Application	Provides access to the ConceptDraw application. Allows to open and close documents, create new documents, libraries and workspace files, import files of various formats and more. An instance of the **Application** object can be retrieved from the **thisApp** global variable.
Character	Provides access to various text properties, such as font, font size, color, style, etc. An instance of the **Character** object can be retrieved from the **Document**, **Style**, **Shape** objects.
Color	An object for working with color. An instance of the **Color** object can be retrieved from the **Character**, **Document**, **Layer**, **Shape**, **Style, TextBlock** objects.
ColorEntry	An object for working with the color palette of the document. An instance of the **ColorEntry** object can be retrieved from the **Document** object.
ConnectDot	This object stores the coordinates of a connection point. An instance of the **ConnectDot** object can be retrieved from the **Shape** object.

ControlDot	Provides access to the properties of a shape's control handle. An instance of the **ControlDot** object can be retrieved from the **Shape** object.
CustomProp	Stores additional information about the shape. An instance of the **CustomProp** object can be retrieved from the **Shape** object.
DataSource	The facility is designed to link the specified object's properties ConceptDraw data at the source. An instance of the **DataSource** object can be obtained using methods of the**Shape.**
DataSourceValue	The facility is designed to provide access to the fields of **Data** table object parameters (shape). The object instance **DataSourceValue** can be obtained using methods of the**Shape.**
Document	Provides access to the contents and properties of the document, opened in ConceptDraw. An instance of the **Document** object can be retrieved by using methods and properties of the **Application** object.
DPoint	A service object which stores point's coordinates, used in coordinate transformations.
DRect	Service object used to store coordinates of a rectangle and containing methods for working with it.
Geometry	The **Geometry** object is used to control the properties of a shape's geometry. It allows to modify the geometry properties, which affect the way the geometry looks. An instance of the **Geometry** object can be retrieved by using the methods of the **Shape** object.
HyperLink	Provides access to the hyperlink properties of the document. Hyperlinks are stored in the document and allow to link shapes of the document to other shapes, pages of the document, other files or URLs. An instance of the **Hyperlink** object can be retrieved by using the methods of the **Document** object.
Layer	Controls properties of a document's layers. You can use layers to organize related objects in the document. An instance of the **Layer** object can be retrieved from the **Document** object.
Library	Provides access to the contents and properties of a library, open in ConceptDraw. Allows to view and edit the contents of the library. An instance of the **Library** object can be retrieved by using the properties and methods of the **Application** object.
Master	Provides access to the properties and contents of a library object (master object). An instance of the **Master** object can be retrieved by using the methods of the **Library** object.
Menu	Provides access to the user-defined menu of the application or document. Is used together with the **MenuItem** object to organize multi-level structure of the user-defined menu in ConceptDraw. An instance of the **Menu** object can be retrieved from the following objects: **Application**, **Document**, **Menu**, **MenuItem**.

MenuItem	Provides access to the contents and properties of a user-defined menu item of ConceptDraw. A menu-item can be associated with a procedure or contain a submenu. An instance of the **MenuItem** object can be retrieved by using the methods of the **Menu** object.
Page	Is used for controlling and accessing the contents of a document page. By using the methods of the **Page** object, you can get access to the existing shapes on the page or create new shapes. An instance of the **Page** object can be retrieved by using the methods and properties of the **Document** object.
Paragraph	Provides access to various paragraph properties of text. An instance of this object can be retrieved from the **Document**, **Style**, **Shape** objects.
ServObj	Used to control various properties of service objects, such as guides. Service objects can be located as on a page, as in a separate group. Service objects carry out auxiliary functions. In Elementicular, a guide line can be used for aligning shapes. An instance of the **ServObj** object can be retrieved from the **Page** and **Shape** objects.
Shape	The **Shape** is used for controlling and obtaining information about a shape in a ConceptDraw document. Provides access to virtually all elements and characteristics of a shape. **Shape** objects can exist on the pages of the document, in libraries, inside groups. An instance of the **Shape** object can be retrieved by using the methods and properties of the following objects: **Page**, **Shape**, **ServObj**, **Master**, **Window**.
Style	Provides access to various style properties of a ConceptDraw document. An instance of the **Style** object can be retrieved by using methods of the **Document** object.
TabStop	Provides access to tabulation properties. An instance of the **TabStop** object can be retrieved from the **TextBlock** object.
TextBlock	Provides access to various text block properties, such as vertical and horizontal alignment and other. An instance of the **TextBlock** object can be retrieved from the **Document**, **Style**, **Shape** objects.
Variable	A service object. You may need variables when several different fields use result of the same calculations. So, the additional variables can be used to store the results. You may also use the additional variables to store various object parameters, which you're working with, so that you don't have to refer to them. An instance of this object can be retrieved from the **Shape** object.
Window	Is used for controlling and obtaining information about the state of the library window of ConceptDraw. An instance of the **Window** object can be retrieved from the following objects: **Application**, **Document**.

Character Object

Text in ConceptDraw shapes can consist of blocks (blocks of characters, sequence of characters) with uniform formatting attributes: font, font size, color, style, etc. The **Character** object provides access to various properties of such a text block.

Properties

Color	Read-only. The color of the characters.
Count	Read-only. The number of characters in this character block.
Font	The font of character block.
Hyperlink	The ID of the hyperlink associated with this character block.
Language	Text encoding.
Pos	Position relative to text baseline (subscript, superscript).
Size	Font size.
Spacing	Spacing between characters.
Style	The font style (bold, italic, underline, etc).

Remarks

Character blocks are stored in the ConceptDraw shape and describe the way the shape's text is displayed. The **Shape** object contains a number of methods for working with the character block collection. Each character block describes the number of characters, defined by **Count**. The properties defined by the character block are applied to the text of the shape according to the order of blocks in the character block collection.

An instance of the **Character** object can be retrieved by using the same properties and methods:
Document object: DefCharacter property.
Shape object: Character method.
Style object: Character property.

To create a new character block in a shape with with the specified parameters, you can use the following methods of the **Shape** object: SetCharColor method, SetCharFont method, SetCharHyperlink method, SetCharLanguage method, SetCharPos method, SetCharSize method, SetCharSpacing method.

See Also Color object, Document object, Paragraph object, Shape object, Style object

ColorEntry Object

ColorEntry Object

An object for working with color palette element.

Properties

isRGB	Read-only. Returns **True** if the color scheme is RGB. Otherwise returns **False**.
isCMYK	Read-only. Returns **True** if the color scheme is CMYK. Otherwise returns **False**.
isTransparent	**True** if the color is transparant, otherwise **False**.
Red	The color's red component in the RGB scheme.
Green	The color's green component in the RGB scheme.
Blue	The color's blue component in the RGB scheme.
Cyan	The color's cyan component in the CMYK scheme.
Magenta	The color's magenta component in the CMYK scheme.
Yellow	The color's yellow component in the CMYK scheme.
Black	The color's black component in the CMYK scheme.

Methods

SetRGB	Sets an RGB color.
SetCMYK	Sets a CMYK color.

See Also Color object, Document object

Color Object

Color Object

An object for working with color. An instance of this object can be retrieved from the Document Object, Style Object, Shape Object, Character Object, TextBlock Object, Layer Object objects.

Properties

isIndex	Read Only. Returns a Boolean value. If the object has indexed color, returns TRUE. Otherwise returns FALSE.
isRGB	Read Only. Returns a Boolean value. If the object has RGB color, returns TRUE. Otherwise returns FALSE.
isCMYK	Read Only. Returns a Boolean value. If the object has CMYK color, returns TRUE. Otherwise returns FALSE.
isTransparent	Gets or sets a Boolean value. If the object has transparent color, returns TRUE. Otherwise returns FALSE.
Index	Gets or sets an Integer value. Is the index of the color in the palette.
Red	Gets or sets an Integer value. Represents the red component of RGB color.
Green	Gets or sets an Integer value. Represents the green component of RGB color.
Blue	Gets or sets an Integer value. Represents the blue component of RGB color.
Cyan	Gets or sets an Integer value. Represents the cyan component of CMYK color.
Magenta	Gets or sets an Integer value. Represents the magenta component of CMYK color.
Yellow	Gets or sets an Integer value. Represents the yellow component of CMYK color.
Black	Gets or sets an Integer value. Represents the black component of CMYK color.

Methods

SetRGB	Sets an RGB color.
SetCMYK	Sets a CMYK color.
GetRed	Returns the Integer value of the red color component, regardless of the color scheme of the Color object.
GetGreen	Returns the Integer value of the green color component, regardless of the color scheme of the Color object.
GetBlue	Returns the Integer value of the blue color component, regardless of the color scheme of the Color object.
GetCyan	Returns the Integer value of the cyan color component, regardless of the color scheme of the Color object.
GetMagenta	Returns the Integer value of the magenta color component, regardless of the color scheme of the Color object.

GetYellow	Returns the Integer value of the yellow color component, regardless of the color scheme of the Color object.
GetBlack	Returns the Integer value of the black color component, regardless of the color scheme of the Color object.

See Also	Character object, ColorEntry object, Document object, Layer object, Shape object, Style object, TextBlock object

ConnectDot Object

ConnectDot Object

Object for storing coordinates of a connection point. Connection Points are associated with an object. They indicate locations in which other objects can be glued to it. An instance of this object can be retrieved from the Shape object.

Properties

Name	Description
X	The X-coordinate of the connection point in the coordinate system of the shape to which it belongs.
Y	The Y-coordinate of the connection point in the coordinate system of the shape to which it belongs.

Example

```
Dim MyConnectDot as ConnectDot, MyShape As Shape
MyShape = thisDoc.ActivePage.DrawRect(50,50,500,500)    ' Create Shape object
MyConnectDot = MyShape.AddConnectDot()
MyConnectDot.X = 100
MyConnectDot.Y = 50
MyShape.PropertyChanged(CDPT_CONNECT_X)
MyShape.PropertyChanged(CDPT_CONNECT_Y)
```

See Also Shape object

ConceptDraw Objects Constants

ConceptDraw access Objects Constants

Import / Export Constants

These constants are used in the Import/Export methods (such as Import method, Export method).

Constant	Value	Import	Export	Description
cdf_UNKNOWN	0	-	-	Means unknown format of file.
cdf_CDD	1	Yes	Yes	ConceptDraw V document file format.
cdf_CDT	2	Yes	Yes	ConceptDraw V template file format.
cdf_CDL	3	Yes	Yes	ConceptDraw V library file format.
cdf_CDW	4	Yes	Yes	ConceptDraw V workspace file format.
cdf_CDD1X	5	Yes	Yes	ConceptDraw 1.x document file format.
cdf_CDT1X	6	Yes	Yes	ConceptDraw 1.x template file format.
cdf_CDL1X	7	Yes	Yes	ConceptDraw 1.x library file format.
cdf_CDW1X	8	Yes	Yes	ConceptDraw 1.x workspace file format.
cdf_CDB	9	No	No	ConceptDraw Basic script source file format.
cdf_BMP	10	Yes	Yes	Bitmap file format.
cdf_DIB	11	Yes	Yes	Device-independent bitmap file format.
cdf_DCM	12			
cdf_GIF	13	Yes	Yes	Graphics Interchange format.
cdf_ICO	14	Yes	Yes	Windows icon file format.
cdf_ICON	15	Yes	Yes	Windows icon file format.

cdf_JPEG	16	Yes	Yes	Joint Photographic Experts Group file format.
cdf_JPG	17	Yes	Yes	Joint Photographic Experts Group file format.
cdf_PNG	18	Yes	Yes	Portable Network Graphics file format.
cdf_PCD	19	Yes	Yes	
cdf_PCDS	20			
cdf_PCX	21	Yes	Yes	
cdf_SGI	22	Yes	Yes	
cdf_RAS	23	Yes	Yes	
cdf_SUN	24			
cdf_TGA	25	Yes	Yes	
cdf_ICB	26			
cdf_VDA	27			
cdf_VST	28			
cdf_TIF	29	Yes	Yes	Tag Image file format.
cdf_TIFF	30	Yes	Yes	Tag Image file format.
cdf_WPG	31	Yes	Yes	
cdf_XBM	32	Yes	Yes	
cdf_XPM	33	Yes	Yes	
cdf_PCT	34	Yes	Yes	
cdf_DXF	35	Yes	Yes	
cdf_HTM	36	No	Yes	Hypertext Markup Language file format.
cdf_HTML	37	No	Yes	Hypertext Markup Language file format.
cdf_EPS	38	No	Yes	Encapsulated postscript file format
cdf_CDX	39	Yes	Yes	XML for ConceptDraw file format.
cdf_OUTLINE	40	Yes	Yes	ConceptDraw outline file format. It is text format file.
cdf_FLOWDATA	41	Yes	Yes	ConceptDraw flowdata file format.
cdf_PPT	42	Yes	Yes	MS PowerPoint file format
cdf_EMF	43	Yes	Yes	Enhanced Metafile format.
cdf_WMF	44	Yes	Yes	Windows Metafile format.

cdf_PAL	45			
cdf_SWF	46	No	Yes	Macromedia Flash format.
cdf_PDF	47	No	Yes	
cdf_PSD	48	Yes	Yes	Adobe Photoshop Drawing format.
cdf_VDX	49	Yes	No	Microsoft Visio XML format.
cdf_SVG	50	No	Yes	Scalable Vector Graphic.
cdf_PICT	51	Yes	No	Macintosh PICT.
cdf_CDOCMD	52	Yes	YES	Conceptdraw Office command file format.
cdf_CDLX	53	Yes	Yes	ConceptDraw XML Libraries file format.
cdf_CDTX	54	Yes	Yes	ConceptDraw XML Template file format.

Property Tag Constants

Constant	**Value**
CDPT_WIDTH	1
CDPT_HEIGHT	2
CDPT_ANGLE	3
CDPT_GPINX	4
CDPT_GPINY	5
CDPT_FLIPX	6
CDPT_FLIPY	7
CDPT_LPINX	8
CDPT_LPINY	9
CDPT_BEGINX	10
CDPT_BEGINY	11
CDPT_ENDX	12
CDPT_ENDY	13
CDPT_GEOMETRY_X	14
CDPT_GEOMETRY_Y	15
CDPT_GEOMETRY_A	16
CDPT_GEOMETRY_B	17

CDPT_GEOMETRY_C	18
CDPT_GEOMETRY_D	19
CDPT_GEOMETRY_VISIBLE	20
CDPT_GEOMETRY_FILLED	21
CDPT_TEXTWIDTH	26
CDPT_TEXTHEIGHT	27
CDPT_TEXTANGLE	28
CDPT_TEXTPINX	29
CDPT_TEXTPINY	30
CDPT_TEXTGPINX	31
CDPT_TEXTGPINY	32
CDPT_VALIGN	33
CDPT_TOPMARGIN	34
CDPT_BOTTOMMARGIN	35
CDPT_LEFTMARGIN	36
CDPT_RIGHTMARGIN	37
CDPT_TEXTBKGND	38
CDPT_DEFTABSTOP	39
CDPT_TABALIGN	40
CDPT_TABPOS	41
CDPT_TEXT	42
CDPT_LINEPATTERN	43
CDPT_LINEWEIGHT	44
CDPT_LINECOLOR	45
CDPT_LINEBEGIN	46
CDPT_LINEEND	47
CDPT_LINEENDSIZE	48
CDPT_FILLPATTERN	49
CDPT_FILLPATCOLOR	50
CDPT_FILLCOLOR	51

CDPT_SHADOWPATTERN	52
CDPT_SHADOWPATCOLOR	53
CDPT_SHADOWCOLOR	54
CDPT_LOCKWIDTH	55
CDPT_LOCKHEIGHT	56
CDPT_LOCKMOVEX	57
CDPT_LOCKMOVEY	58
CDPT_LOCKASPECT	59
CDPT_LOCKCALCWH	60
CDPT_LOCKROTATE	61
CDPT_LOCKDELETE	62
CDPT_LOCKBEGIN	63
CDPT_LOCKEND	64
CDPT_LOCKVERTEX	65
CDPT_LOCKFLIPX	66
CDPT_LOCKFLIPY	67
CDPT_SHOWSHAPEHANDLES	68
CDPT_SHOWCONTROLHANDLES	69
CDPT_SHOWALIGNBOX	70
CDPT_NONPRINTING	71
CDPT_RESIZEBEHAVIOUR	72
CDPT_SHOWTEXT	73
CDPT_VARIABLE_X	74
CDPT_VARIABLE_Y	75
CDPT_CONTROL_X	76
CDPT_CONTROL_Y	77
CDPT_CONTROL_XDYN	78
CDPT_CONTROL_YDYN	79
CDPT_CONTROL_XBEHAVIOUR	80
CDPT_CONTROL_YBEHAVIOUR	81

CDPT_CONTROL_COMMENT	82
CDPT_CONNECT_X	83
CDPT_CONNECT_Y	84
CDPT_CHAR_FONT	85
CDPT_CHAR_SIZE	86
CDPT_CHAR_COLOR	87
CDPT_CHAR_STYLE	88
CDPT_CHAR_POS	90
CDPT_CHAR_LANGUAGE	91
CDPT_CHAR_SPACING	92
CDPT_CHAR_HYPERLINK	93
CDPT_PARA_FIRSTIND	94
CDPT_PARA_LEFTIND	95
CDPT_PARA_RIGHTIND	96
CDPT_PARA_HALIGN	97
CDPT_PARA_BEFORESPACING	99
CDPT_PARA_AFTERSPACING	100
CDPT_PARA_LINESPACING	101
CDPT_ACTION_ACTION	102
CDPT_ACTION_MENU	103
CDPT_ACTION_PROMPT	104
CDPT_ACTION_CHECKED	105
CDPT_ACTION_DISABLED	106
CDPT_CUSTOM_LABEL	107
CDPT_CUSTOM_PROMPT	108
CDPT_CUSTOM_TYPE	109
CDPT_CUSTOM_FORMAT	110
CDPT_CUSTOM_VALUE	111
CDPT_CUSTOM_INVISIBLE	112
CDPT_CUSTOM_VERIFY	113

CDPT_CONNECTOBJBEGIN	129
CDPT_CONNECTOBJEND	130
CDPT_CONNECTTYPEBEGIN	131
CDPT_CONNECTTYPEEND	132
CDPT_TEXTFLIPX	133
CDPT_TEXTFLIPY	134
CDPT_LAYER	142
CDPT_HYPERLINK	143
CDPT_DBLCLICK	144
CDPT_DBLCLICKACTION	145
CDPT_ROUNDCORNERS	148
CDPT_CONNECTMODE	149
CDPT_CONNECTORKNEELIMIT	150
CDPT_CONNECTBYPASSGROUPS	151
CDPT_EVENTPAGESREORDER	152
CDPT_EVENTPAGESCOUNT	153
CDPT_EVENTIDLE	154
CDPT_EVENTTIMER	155
CDPT_EVENTLOAD	156
CDPT_CHARPROPEVENT	157
CDPT_EVENTFILENAME	158
CDPT_LINEALPHA	159
CDPT_FILLFOREGNDALPHA	160
CDPT_FILLBACKGNDALPHA	161
CDPT_SHADOWFOREGNDALPHA	162
CDPT_SHADOWBACKGNDALPHA	163
CDPT_TEXTBKGNDALPHA	164
CDPT_CHAR_ALPHA	165
CDPT_FILLTEXTURE	166
CDPT_REGULAR_PROPS_NUM	167

CDPT_STYLED_LINEPATTERN	210
CDPT_STYLED_LINEWEIGHT	211
CDPT_STYLED_LINECOLOR	212
CDPT_STYLED_BEGINARROW	213
CDPT_STYLED_ENDARROW	214
CDPT_STYLED_ARROWSIZE	215
CDPT_STYLED_FILLPATTERN	216
CDPT_STYLED_FILLFOREGND	217
CDPT_STYLED_FILLBACKGND	218
CDPT_STYLED_SHADOWPATTERN	219
CDPT_STYLED_SHADOWFOREGND	220
CDPT_STYLED_SHADOWBACKGND	221
CDPT_STYLED_CHAR_FONT	222
CDPT_STYLED_CHAR_SIZE	223
CDPT_STYLED_CHAR_COLOR	224
CDPT_STYLED_CHAR_STYLE	225
CDPT_STYLED_CHAR_POS	226
CDPT_STYLED_CHAR_SET	227
CDPT_STYLED_CHAR_SPACING	228
CDPT_STYLED_PARA_FIRSTIND	229
CDPT_STYLED_PARA_LEFTIND	230
CDPT_STYLED_PARA_RIGHTIND	231
CDPT_STYLED_PARA_HALIGN	232
CDPT_STYLED_PARA_BEFOREIND	233
CDPT_STYLED_PARA_AFTERIND	234
CDPT_STYLED_PARA_BETWEENLN	235
CDPT_STYLED_VERTICALALIGN	236
CDPT_STYLED_TOPMARGIN	237
CDPT_STYLED_BOTTOMMARGIN	238
CDPT_STYLED_LEFTMARGIN	239

CDPT_STYLED_RIGHTMARGIN	240
CDPT_STYLED_TEXTBKGND	241
CDPT_STYLED_TXTDEFTABSTOP	242
CDPT_STYLED_LINEALPHA	243
CDPT_STYLED_FILLFOREGNDALPHA	244
CDPT_STYLED_FILLBACKGNDALPHA	245
CDPT_STYLED_SHADOWFOREGNDALPHA	246
CDPT_STYLED_SHADOWBACKGNDALPHA	247
CDPT_STYLED_CHAR_ALPHA	248
CDPT_STYLED_TEXTBKGNDALPHA	249
CDPT_DS_DATASOURCE	250
CDPT_DS_REFRESH_TIME	251
CDPT_DS_ACTION	252
CDPT_DS_VALID	253
CDPT_DS_ACTIVE	254
CDPT_DS_DATASOURCE_PATH	255
CDPT_DS_RELIABILITY	256
CDPT_DS_SHOW_WARNINGS	257
CDPT_DS_SHOW_ERRORS	258
CDPT_DSV_NAME	259
CDPT_DSV_VALUE	260
CDPT_DSV_TYPE	261
CDPT_DSV_VISIBLE	262
CDPT_DSV_OBJECT_TYPE	263
CDPT_DSV_SHOW_DIALOG	264
CDPT_LOCKTEXTBOUND	265
CDPT_LOCKGROUP	266
CDPT_LOCKFILL	267
CDPT_LOCKLINE	268
CDPT_RAPIDDRAW	269

CDPT_HIDEINSLIDESHOW	270
CDPT_RD_LIB_NAME	271
CDPT_RD_OBJ_NAME	272
CDPT_RD_ICON_NAME	273
CDPT_RD_LEFT_PLACING	274
CDPT_RD_RIGHT_PLACING	275
CDPT_RD_TOP_PLACING	276
CDPT_RD_BOTTOM_PLACING	277
CDPT_RD_CONNECTOR_TYPE	278
CDPT_RD_CONN_LIB_NAME	279
CDPT_RD_CONN_OBJ_NAME	280
CDPT_RD_AUTO_BALANCE	281
CDPT_RD_SPACING_X	282
CDPT_RD_SPACING_Y	283
CDPT_RD_START_CONN_POINT	284
CDPT_RD_END_CONN_POINT	285
CDPT_RD_SPACING_X_VERT_MOVE	286
CDPT_RD_SPACING_Y_HOR_MOVE	287
CDPT_RAPIDDRAW_OBJECT_BOUND	288
CDPT_RAPIDDRAW_TOP_AUTO_STEP	289
CDPT_RAPIDDRAW_LEFT_AUTO_STEP	290
CDPT_RAPIDDRAW_RIGHT_AUTO_STEP	291
CDPT_RAPIDDRAW_BOTTOM_AUTO_STEP	292
CDPT_RD_OBJECT_DESCRIPTION	293

ControlDot Object

ControlDot Object

The **ControlDot** object serves for controlling and accessing properties of control handles of ConceptDraw shapes. Control handles allow the shape's properties to be modified automatically when the control handle is repositioned. Each control handle has a formula that determines its coordinates with respect the the coordinate system of the shape to which it belongs. For example, you might use a control handle to adjust the roundness of a shape's corners or to reshape an arrow.

Properties

Name	Description
X	The X-coordinate of the control handle in the coordinate system of the shape it belongs to.
Y	The Y-coordinate of the control handle in the coordinate system of the shape it belongs to.
XDyn	The X-coordinate of the end of the line which comes out of the control handle when the latter is repositioned.
YDyn	The Y-coordinate of the end of the line which comes out of the control handle when the latter is repositioned.
XBehaviour	Sets the behavior of the control handle relative to the X axis when the shape is resized.
YBehaviour	Sets the behavior of the control handle relative to the Y axis when the shape is resized.
Comment	The tip that comes up when you move the mouse pointer over the control handle.

Remarks

An instance of the object can be retrieved by using the methods of the **Shape** object.

See Also Shape object

CustomProp Object

CustomProp Object

This object serves for storing and controlling additional information about the shape, defined by the user (custom properties).

Properties

Label	The label (unique name) of the shape's custom property.

Prompt	The tip of the shape's custom property.
Type	The type of the shape's custom property.
Format	The format of the custom property.
Value	The default value.
Invisible	Visible / invisible state.
Verify	Verify / not verify state.

Remarks

An instance of the **CustomProp** object can be retrieved by using the following methods of the **Shape** object: AddCustomProp method, CustomProp method, CustomPropByLabel method.

See Also Shape object

DataSourceValue Object

DataSourceValue Object

Object **DataSourceValue** designed to provide access to fields table **Data** parameters of the object (shape). Using the properties of this object, you can receive, edit and use data tables **Data.**

Properties

Name	Data from the **Data** Section **Name** Table of parameters of the object.
Value	These sections of the **Value Data** Table parameters of the object.
Type	Data from Table **Data Type** section of the object parameters.
Visible	These sections of the **Visible Data** table parameter object.

Remarks

The object instance **DataSourceValue** can be obtained using methods of the **Shape.**

See Also Shape object, DataSource object, AddDSValue method, DSValue method, DSValueEl method

DataSource Object

DataSource Object

A **DataSource** object is designed to link the specified object properties (shapes) with the data source. Using properties and methods, we can obtain and modify the mode of a data source.

Properties

Action	The action to be performed in case of receiving new data from the source.
Active	Provides a start or stop the update process data from the source.
DataSource	The relative or full path to the data source.
Refresh	The time interval in seconds, through which the update data from the source.
ShowErrors	Determines whether to display the appropriate icon when an error occurs while working with a data source.
ShowWarnings	Determines whether to display the appropriate icon in the event of the comments in the process of working with a data source.
Timeout	The time interval in seconds over which will be by appropriate icon in the event of an error when updating the data from the source.

Methods

ColCount	Returns the maximum number of columns in the search for all rows in a table view CSV file data source.
RowCount	Returns the number of non-empty string, ie rows that contain data in a tabular representation of the CSV file data source.

Remarks

An instance of the **DataSource** object can be obtained using methods of the **Shape.**

See Also Shape object, DataSourceValue object, AddDataSource method, DataSource method

Document Object

Document Object

The **Document** object is used to control a ConceptDraw document and its contents. It includes all the necessary properties and methods for working with the contents of the document: pages, layers, colors, styles, hyperlinks, color palette, various document settings, etc.

Properties

Name	Description
ActiveLayer	Active layer **ID** (the **ID** property).
ActivePage	Read-only. Active page of the document.
ActiveView	Read-only. Active window of the document (represents an instance of the active document view).
Author	Describes the document's author.
Company	Describes the company which created the document.
CustomMenu	Read-only. Returns the user-defined menu of the document.
DefCharacter	Read-only. The sequence of characters, assigned to the new shape's text by default.
DefFillColor	Read-only. Default fill color for new shapes.
DefFillPatColor	Read-only. Default fill pattern color, applied by default to every new shape.
DefFillPattern	The type of fill pattern, applied by default to every new shape.
DefParagraph	Read-only. Paragraph properties, applied by default to every new shape.
DefPenColor	Read-only. Default line color for new shapes.
DefPenPattern	The type of line pattern, applied by default to every new shape.
DefPenWeight	Default line weight for new shapes.
DefShadowColor	Read-only. Default shadow color for new shapes.
DefShadowPatColor	Read-only. The shadow fill color, applied by default to every new shape.
DefShadowPattern	Read-only. The type of shadow pattern, applied by default to every new shape.
DefStyle	Default style, applied to all new shapes.
DefTextBlock	Read-only. The text block, assigned by default to every new shape.
Desc	Detailed description of the document.
FlowAroundObjects	A flag that specifies whether smart connectors should flow around shapes on their way.
FullName	Read-only. Returns the full filename of the document.
LineJumpOrient	Orientation of smart connector's crossings in the document.
LineJumpSize	The size of smart connector's crossings in the document.
LineJumpType	The type of smart connector's crossings in the document.

MaxNumberOfLegs	The maximum number of Smart Connector's legs.
MinDistToShapes	The minimum possible distance between a a smart connector and any other shapes on the page, on which the smart connector is located.
Name	Document file name.
PageSizeX	Document page width. It's specified in internal units of ConceptDraw (**InternalUnit**).
PageSizeY	Document page height. It's specified in internal units of ConceptDraw (**InternalUnit**).
PassThroughGroups	A flag that specifies whether the smart connectors in the document should flow around the whole group (**False**) or individual shapes inside the group on the connector's way (**True**).
Path	Path to the document filename.
Scale	Scale, set in the document.
ShadowOffsetX	Horizontal shadow offset.
ShadowOffsetY	Vertical shadow offset.
SnapSensitivity	Current snap sensitivity for the document.
SplineSmooth	Current spline smoothness for the document.
Subj	Brief description of the document.
Title	Title of the document.
UnitIndex	Units of measure of the document.

Methods

Name	Description
AddHyperlinkToDocument	Adds a hyperlink to a ConceptDraw document.
AddHyperlinkToFile	Adds a hyperlink to a local file.
AddHyperlinkToPageShape	Adds a hyperlink to a shape or a page of the current document.
AddHyperlinkToURL	Adds a hyperlink to a URL.
AddLayer	Adds a new layer to the document's layer collection.
AddPage	Adds a new page to the document.
AddStyle	Adds a new style to the document's style collection.
ColorEntry	Returns a color from the color table by its index in the color collection of the document.
ColorsNum	Returns the number of colors in the color table of the document.
EndRebuild	Informs the ConceptDraw engine about the termination of modifying properties of the shapes of the document.

Export	Exports the document to the specified file format.
FindFontByName	Returns the font's index in the document's font collection.
FindPage	Returns the page's index in the document's page collection.
FindStyle	Returns the style's index in the document's style collection.
FirstView	Returns the first window in the document's window collection.
FontName	Returns the font name by the specified font index in the document's font collection.
FontsNum	Returns the number of fonts in the document's font collection.
Hyperlink	Returns a hyperlink by the specified index in the document's hyperlink collection.
HyperlinkByID	Searches for a hyperlink by the specified ID in the document's hyperlink collection.
HyperlinksNum	Returns the number of hyperlinks in the document's hyperlink collection.
Layer	Returns a layer by the specified index in the document's layer collection.
LayerByID	Searches for a layer by its ID (the **ID** property) in the document's layer collection.
LayerByName	Searches for a layer by its name (the **Name** property) in the document's layer collection.
LayersNum	Returns the number of layers in the document's layer collection.
MoveShapeToGroup	Moves the object (shape) to the group.
MoveShapeToPage	Moves the object (shape) in a specific position on another page.
NextView	Returns the next window from the document's window collection.
Page	Returns a page by its index in the document's page collection.
PageByID	Searches for a page by its ID (the **ID** property) in the document's page collection.
PagesNum	Returns the number of pages in the document.
RemoveLayer	Removes a layer by its index in the document's layer collection.
RemoveLayerByID	Removes a layer with the specified ID (the **ID** property) from the document.
RemovePage	Removes a page by its index in the document's page collection.
RemovePageByID	Removes a page with the specified ID (the **ID** property) from the document.
RemoveStyle	Removes a style by its index in the document's style collection.
RemoveStyleByName	Removes a style with the specified name (the **Name** property) from the document's style collection.

RemoveUnusedHyperlinks	Removes unused hyperlinks from the document's hyperlink collection.
RenameStyle	Renames a style.
ReorderPage	Reorders pages in the document's page collection. The page being moved is defined by the index in the page collection of the document.
ReorderPageByID	Reorders pages in the document's page collection. The page being moved is defined by its ID (the **ID** property) in the page collection of the document.
Save	Saves the document.
SaveAs	Saves the document with the specified parameters.
SetActivePage	Sets an active page in the document by its index.
SetActivePageByID	Sets an active page in the document by its **ID**.
SetActiveView	Activates the specified view of the document.
StartRebuild	Informs the ConceptDraw engine about the beginning of modifying properties of the shapes of the document.
Style	Returns a style by the specified index in the document's style collection.
StyleByName	Searches for a style with the specified index in the document's style collection.
StylesNum	Returns the number of styles in the document's style collection.
UpdateAllViews	Redraws all windows of the document.
ViewByID	Searches for the window with the specified ID (the ID property) in the view collection of the document.
ViewsNum	Returns the number of views of the current document.

Remarks

Documents, opened in the application, can be located in the application only. The **Application** object has a number of methods for controlling ConceptDraw documents, opened in the application. However, methods and properties of the **Application** object are not the only means to retrieve an instance of the **Document** object. Other objects that belong to or are associated with the document can refer to it.

AElement from controlling all the collections stored in a ConceptDraw document (pages, layers, hyperlinks, styles, color palette) the **Document** object provides access to all windows which display the contents of the document, and also allows to control re-calculation of properties of the document's shapes (StartRebuild, EndRebuild), redraw the document's windows (UpdateAllViews), etc.

An instance of the **Document** object can be retrieved from the following methods and properties:
Application object: ActiveDoc property, CreateNewDoc method, Doc method, DocByName method, FirstDoc method, Import method, NextDoc method, OpenDoc method.

Page, **ServObj**, **Shape** objects: Document property.
Window object: Document property.

Also an instance of the **Document** object can be retrieved by using the **thisDoc** global variable, pre-defined at document script, page script, shape script levels. The **thisDoc** variable in the document-level script returns the document which script is being executed when the variable is referred to. For the page/shape-level script it returns the document, to which belongs the page/shape, which script is being executed.

See Also	Application object, Hyperlink object, Layer object, Page object, Style object, Window object

DPoint Object

DPoint Object

Service object used to store coordinates of a point, used in coordinate transformations.

Properties

X	Gets or sets a Double value, representing the X coordinate of the point.
Y	Gets or sets a Double value, representing the Y coordinate of the point.

Methods

Equal	Copies properties of an object of the same type.

Remarks

Note, that an instance of the object should be created prior to using it (before or after its declaration). The example below demonstrates how to do it:

Example

```
' The New keyword is used to create a new instance of the object
Dim MyObject as new DPoint
' one more method for creating an instance of the object
Dim MySecondObject as DPoint
set MySecondObject = new DPoint
```

See Also	DRect Object, Dim Statement , Set Statement, LPtoGP Method, LPtoWP Method, WPtoLP Method

DRect Object

DRect Object

A service object used to describe and perform various operations with a rectangle.

Properties

Name	**Description**
left	Gets or sets a Double value, representing the coordinate of the leftmost point of the rectangle.
top	Gets or sets a Double value, representing the coordinate of the top point of the rectangle.
right	Gets or sets a Double value, representing the coordinate of the rightmost point of the rectangle.
bottom	Gets or sets a Double value, representing the coordinate of the bottom point of the rectangle.

Methods

Name	**Description**
Equal	Copies properties of an object of the same type.
SetRect	Sets left, top, right, bottom points of an object.
InflateRect	Extends the rectangle by the X and Y axis, calculates new coordinates of the object.
DeflateRect	Shrinks the rectangle by the X and Y axis, calculates new coordinates of the object.
GetWidth	Returns the width of the rectangle.
GetHeight	Returns the height of the rectangle.
isEmpty	Returns TRUE if the square of the rectangle equals to 0. Otherwise returns FALSE.
PtInRect	Returns TRUE if the specified point's coordinates fall within the rectangle's bound. Otherwise returns FALSE.

OffsetRect	Moves the rectangle by the X and Y axis, calculates new coordinates of the object.
SetRectEmpty	Resets object properties to zero.
UnionRect	Calculates the coordinates of the minimum rectangle enough to encompass two specified rectangles. Returns FALSE if the specified rectangles are empty, otherwise returns TRUE.
IntersectRect	Calculates the coordinates of the rectangle, resulting from the intersection of two specified rectangles. If such rectangle exists, returns TRUE, otherwise returns FALSE.
NormalizeRect	Resets object properties to defaults.

Remarks

The **DRect** object is not used in the object hierarchy of ConceptDraw Basic and is used exclusively to facilitate working with rectangles thanks to the methods, implemented in the **DRect** object**.**

Example

```
' The New keyword is used to create a new instance of the object
Dim MyObject as new DRect
' one more method for creating an instance of the object
Dim MySecondObject as DRect
set MySecondObject = new DRect
```

See Also DPoint Object, Dim Statement , Set Statement

Geometry Object

Geometry Object

The **Geometry** object is used to control the properties of the shape's geometry. It allows to modify the geometry properties, which affect the way the geometry looks.

Properties

Name	**Description**

Visible	A flag, that indicates whether the geometry is visible.
Filled	A flag, that indicates, whether to fill the area, enclosed by the geometry.

Methods

SegmentsNum	Returns the number of segments in geometry.

Remarks

A shape includes one or more geometries, containing elementary segments, which make up the shape. A geometry can be visible, invisible, filled or not filled. Normally, when you draw a shape using the drawing tools in ConceptDraw, a new geometry is added when you add new segments to a shape, created earlier.

An instance of the **Geometry** object can be retrieved by using the following methods of the **Shape** object: AddGeometry method, Geometry method.

See Also Shape object

HyperLink Object

HyperLink Object

This object describes a hyperlink in a ConceptDraw document. A **Hyperlink** object enables you to access and manipulate the properties and behavior of a a hyperlink.

Properties

ID	Read-only. Hyperlink's ID.
LinkType	Read-only. Hyperlink's type.
Address	Read-only. A string with the address to which the hyperlink navigates.
LocalPath	Read-only. A flag that specifies whether the **Address** is a relative (local) path.
PageID	Read-only. ID of the document page, to which the hyperlinks points.
ShapeID	Read-only. ID of the shape, to which the hyperlink points.

Remarks

Hyperlink objects are stored in the hyperlink collection of the **Document** object. You can create a hyperlink using the ConceptDraw interface, or using the following methods of the **Document** object: AddHyperlinkToDocument, AddHyperlinkToFile, AddHyperlinkToPageShape, AddHyperlinkToURL. An instance of the **Hyperlink** object can be retrieved from the document's hyperlink collection with the help of the following methods of the **Document** object: Hyperlink, HyperlinkByID.

See Also Document object, Page object, Shape object

Layer Object

Layer Object

You can use layers to organize related objects in the document. An instance of the object can be retrieved from the Document Object.

Properties

ID	Read Only. The layer ID.
Name	The layer's name.
Visible	The visibility flag.
Locked	Flag that specifies whether the layer can be edited.
Printable	Flag that specifies whether the layer can be printed.
Colored	Flag that specifies whether the layer is colored.
Color	Read-only. The color of shapes on the colored layer.

See Also Color Object, Document Object, ServObj Object, Shape Object

Library Object

Library Object

The **Library** object is used to get information and control the contents of a ConceptDraw library. The methods of the **Library** object allow to modify the description of the library, save the library and control master objects , stored in the library.

Properties

Name	The library file name.
FullName	Read-only. Full library filename, including the path.
Path	Path to the library file (without filename).
Title	The library title.
Author	Describes the author of the library.
Subj	Brief description of the library.
Company	Describes the company which created the library.
Desc	Contains detailed description of the library.

Methods

Save	Saves the library.
SaveAs	Saves the library with a file name.
AddMaster	Adds a new master object (library object) to the library.
FindMaster	Searches for a specified master object (library object) in the library.
Master	Returns an existing master object (library object) by its index in the library's collection of master objects.
MasterByName	Searches for a master object (library object) by the specified name.
MastersNum	Returns the number of master objects (library objects) in a library.
RemoveMaster	Removes a master object (library object) from a library by its index in the library's collection of master objects.
RemoveMasterByName	Removes a master object (library object) from the library by the specified name.

Remarks

The **Application** object has a number of methods for working with libraries opened in the application. However, other objects can also refer to an open library. An instance of the **Library** object can be retrieved by using the following methods and properties:
Application object: ActiveLib property, CreateNewLib method, Lib method, LibByName method, OpenLib method.
Window object: Library property, Lib method, LibByName method.

See Also Application object, Master object, Shape object

Master Object

Master Object

The **Master** object represents a library object (master object). A library object is an item stored in a library and that contains the following data describing the shape: the name of the library object, the description of the object, the icon and the shape itself.

Properties

Name	Description
Shape	Read-only. The shape, contained in the master object.
Name	The name of the library shape.
Prompt	A brief description of the master object.

Methods

Name	Description
Equal	Makes the library shape equivalent to the specified library shape.
SetShape	Sets the specified shape as the shape, contained in the given master object.
SetIcon	Sets the image, contained in the specified file as the icon for the master shape.

Remarks

An instance of the **Master** object can be retrieved by using the following methods of the **Library** object: AddMaster method, Master method, MasterByName method.

See Also Library object, Shape object

MenuItem Object

MenuItem Object

Represents a menu item of the ConceptDraw user-defined menu.

Properties

Type	Read-only. The menu item type.
CmdID	Read-only. The menu item ID.
Caption	The name of the menu item.
Prompt	The prompt for the menu item.
Enabled	A flag that specified whether the menu item is enabled or disabled.
Checked	A flag that specified whether the menu item is checked.
Parent	Read-only. The parent menu for the menu item.
SubMenu	Read-only. The submenu for the menu item.
OnCmdModule	Read-only. The name of the module associated with the menu item.
OnCmdSub	Read-only. The name of a procedure to process the menu item.
OnCmdArgs	The argumets string wich passed to the processing procedure.

Methods

SetCmdProcessing	Sets a procedure for processing the menu item.

Remarks

An instance of the **MenuItem** object can be retrieved by using the following methods of the **Menu** object:AddMenuItem method, MenuItem method, MenuItemByCmdID method.

See Also	AddMenuItem method, MenuItem method, MenuItemByCmdID method, Menu object

Menu Object

Menu Object

The Menu object is used to control the user-defined menu of the ConceptDraw application.

Properties

CmdID	Read-only. The menu ID.
Caption	The name of the menu.
Prompt	The prompt for the menu.
Enabled	A flag that specifies whether the menu is enabled or disabled.
Parent	Read-only. The parent menu for the menu.

Methods

AddMenuItem	Adds a menu item.
MenuItem	Returns a menu item by its index in the menu collection.
MenuItemByCmdID	Returns a menu item by its ID (the **CmdID** property).
MenuItemsNum	Returns the number of items in the menu.
RemoveMenuItem	Removes a menu item by its index in the menu collection.
RemoveMenuItemByCmdID	Removes a menu item by its ID (the **CmdID** property).
RemoveAll	Removes all menu items.
FindMenuItem	Searches for a menu item among the items of the menu.

Remarks

An instance of the **Menu** object can be retrieved by using the following methods and properties:
Application object: CustomMenu property,
Document object: CustomMenu property,
Menu object: Parent property,
MenuItem property: Parent property, SubMenu property.

Example

An aplication level or a document level script can add items to the custom menu of the document and process them by using its own procedures. Below is an example of such program:

```
' Definition of procedure
Sub MenuItem1_CmdProc(cmdArgs As String)
       Trace "MenuItem1 : " & cmdArgs
        ' ...
        ' ...
End Sub
Dim mi As MenuItem
' Enable Document custom menu
thisDoc.CustomMenu.Caption = "My Doc menu"
' Add menu item
set mi = thisDoc.CustomMenu.AddMenuItem(0)
' Set menu item caption
mi.Caption = "Item 1"
mi.OnCmdArgs = "Args string from menu item"
' Set processing procedure
mi.SetCmdProcessing("MenuItem1_CmdProc")
' Suspends execution
Stop
```

See Also Application object, Document object, MenuItem object

Page Object

Page Object

The **Page** object is used to get information about and control the contents of a ConceptDraw document's page. The methods and properties of the **Page** object allow to create on the page simple shapes, groups, service objects and other objects, as well as control existing objects.

Properties

BackPageID	The background page ID (the **ID** property).
Document	The document which contains this page.
ID	The page ID.

IsBackground	The flag that specifies whether this page can be a background page.
Name	The page name.

Methods

ArcTo	Creates an arc.
BeginShape	Returns the current shape, being edited (the current Basic shape).
ConvertToGroup	Converts a Vector Picture to a group.
ConvertToVFPicture	Converts a shape to a Vector Picture.
DoForConnected	Causes BASIC procedure with an appropriate title for each of the objects that are attached (either directly or through other objects) to the object with the specified identifier.
DrawConnector	Creates a connector.
DrawGroup	Creates a group.
DrawGuide	Creates the Guide service object.
DrawLine	Creates a line.
DrawOval	Creates an ellipse.
DrawRect	Creates a rectangle.
DrawSector	Creates a sector of a circle or ellipse.
DrawSmartConnector	Creates a Smart Connector.
DrawStamp	Draws a copy of the specified shape with the specified size and position, same as the Stamp tool in ConceptDraw.
DrawStampSelection	Draws copies of all selected shapes with the specified size and position, same as the Stamp tool in ConceptDraw.
DropStamp	Creates a copy of the shape, same as the Stamp tool in ConceptDraw, preserving the size of the original shape.
DropStampSelection	Creates copies of all selected shapes, same as the Stamp tool in ConceptDraw, preserving the size of the original shapes.
EndShape	Notifies when creation of the shape is finished.
GetShapeByName	Searches for the object (shape) for a given name in the position stranitse.Returns found object (shape) in a collection of objects (shapes).
InsertPicture	Inserts a picture from a file onto the page.
LineTo	Creates a line in the current Basic shape for this page.
MoveTo	Specifies the position of the current point of the shape, used for creating it.
RemoveAllServObjs	Removes all service objects on the page.
RemoveAllShapes	Removes all shapes on the page.

RemoveServObj	Removes a service object by its index in the service object collection of the document.
RemoveServObjByID	Removes a service object by its ID (the **ID** property).
RemoveShape	Removes a shape by its index in the shape collection of the page.
RemoveShapeByID	Removes a shape by its ID (the **ID** property).
ReorderServObj	Moves the service object to the specified position in the service object collection of the page. The service object to be repositioned is indicated by its index.
ReorderServObjByID	Moves the service object to the specified position in the service object collection of the page. The service object to be repositioned is indicated by its ID (the **ID** property).
ReorderShape	Moves the shape to the specified position in the shape collection of the page. The shape to be repositioned is indicated by its index.
ReorderShapeByID	Moves the shape to the specified position in the shape collection of the page. The shape to be repositioned is indicated by its ID (the **ID** property).
ServObj	Returns a service object by its index in the service object collection of the page.
ServObjByID	Returns a service object by the specified unique number (the **ID** property) of the service object.
ServObjsNum	Returns the number of service objects on the page.
Shape	Returns a shape by its index in the shape collection of the page.
ShapeByID	Returns a shape by its unique number (the **ID** property).
ShapeBySubID	Returns a shape by its unique number (the **SubID** property).
ShapesNum	Returns the number of shapes on the page.
SplineStart	Adds the Spline start segment to the shape.
SplineTo	Draws a spline in the shape.

Remarks

Pages can be stored only inside a ConceptDraw document. Each ConceptDraw document contains its own page collection which can be controlled by using the methods and properties of the **Document** object. However, the **Document** object is not the only way to get an instance of the **Page** object, as various objects (service objects and regular shapes) can reference to the page to which they belong.

Methods for drawing shapes on the page are similar to those used for drawing shapes in a group, as the page is in fact a parent object for shapes, as the group is. The coordinate system of the page is at the highest level and is referred to as "global" in ConceptDraw. The methods of the **Page** object allow to work with all types of ConceptDraw objects that can exist on a document page. All objects that belong to a page are stored in two collections - regular shapes and service objects. The **Page** object has corresponding groups of methods for working with these collections (see above).

An instance of the **Page** object can be retrieved by using the following methods and properties:
Document object: ActivePage property, AddPage method, Page method, PageByID method.
SerbObj, **Shape** objects: Page property.
Window object: Page property.

Also an instance of the **Page** object can be retrieved by using the **thisDoc** global variable, pre-defined at the page and shape script levels. The **thisDoc** variable in the page-level script returns the page which script is being executed when the variable is referred to. For the shape-level script it returns the page, to which belongs the shape, which script is being executed.

See Also Document object, ServObj object, Shape object

Paragraph Object

Paragraph Object

The **Paragraph** object serves for controlling a ConceptDraw shape's text paragraph. It represents a block of text that ends with a line feed symbol. Shape's text may have several paragraphs. A paragraph contains the parameters for alignment, indents, line spacing and other of the text which it contains.

Properties

Name	Description
AfterSpacing	The distance between this paragraph and the one below.
BeforeSpacing	The distance between this paragraph and the one above.
Count	Read Only. Returns the number of characters in the paragraph.
FirstInd	The first line indent value.
HAlign	The horizontal alignment type for the paragraph.
LeftInd	The distance all lines of text in a paragraph are indented from the left margin of the text block.
LineSpacing	The distance between one line of text and the next.

RightInd	The distance all lines of text in a paragraph are indented from the right margin of the text block.

Remarks

Paragraphs are stored in a ConceptDraw shape and describe the appearance of the shape's text. The **Shape** object contains a number of methods for working with its own paragraph collection. Each paragraph describes the number of symbols specified in the **Count** property. Paragraph properties are applied to the shape's text in the same order as the paragraphs are located in the paragraph collection of the shape.

An instance of the **Paragraph** object can be retrieved by using the following properties and methods:
Document object: DefParagraph property.
Shape object: Paragraph method.
Style object: Paragraph property.

To create a new paragraph with specified parameter in a shape, use the following methods of the **Shape** object: SetParaAfterSpacing method, SetParaBeforeSpacing method, SetParaFirstInd method, SetParaHAlign method, SetParaLeftInd method, SetParaLineSpacing method, SetParaRightInd method.

See Also	Character object, Document object, Shape object, Style object, TextBlock object

ServObj Object

ServObj Object

The **ServObj** object describes properties of a ConceptDraw service object - such as a guide line. By using properties of **ServObj**, you can change the angle of the guide, position of its rotation center, name and description, find out to which group, document or page this service object belongs, etc.

Properties

Angle	The angle to which the service object (guide) is rotated with respect to its rotation center.

Desc	A brief description of the service object.
Document	Read-only. The document, which contains the service object.
GPinX	The X coordinate of the service object's rotation center in the global coordinate system - that is, in the coordinate system of its parent group/page.
GPinY	The Y coordinate of the service object's rotation center in the global coordinate system - that is, in the coordinate system of its parent group/page.
ID	Read-only. The service object ID.
Layer	The layer on which the service object lies.
Name	The name of the service object.
ObjType	Read-only. The service object type.
Page	Read-only. The page, which contains the service object.
Parent	Read-only. The parent group of the service object.
SubID	Read-only. The ID of the service object inside its parent shape (group).

Methods

Equal	Copies all properties and contents of the specified service object to this service object.
GetDoubleProperty	Returns the value of a **Double** type property.
GetPropertyFormula	Returns the property's formula in the form of a string.
GetStringProperty	Returns the value of a **String** type property
IsDefaultFormula	Returns **True** if the specified property of the service object has a formula that is marked as default. Otherwise returns **False**.
IsNullFormula	Returns **True** if the specified property of the service object has no formula. Otherwise returns **False**.
PropertyChanged	Tells the ConceptDraw engine that the value of the specified table property has been changed and the formulas of the dependent properties must be re-calculated.
RecalcProperty	Tells the ConceptDraw engine that the value of the specified table property is to be re-calculated using its table formula.
SetDefaultFormula	Creates a default formula for the specified table property of the shape.
SetDoubleProperty	Sets a value of the specified **Double** type table property.
SetNullFormula	Removes the formula from the specified table property of the shape.

SetPropertyFormula	Sets a table formula for the specified table property of the shape.
SetStringProperty	Sets a value of the specified **String** type table property.

Remarks

Service objects can be located inside a group or by themselves on a ConceptDraw document page. Each page and group of a ConceptDraw document contain their own collections of service objects and have corresponding methods for working with service objects. An instance of the **ServObj** object can be retrieved by using the following methods of the **Page** and **Shape** objects: ServObj method, ServObjByID method.

To draw a new guide on a page or in a group you can use the DrawGuide method.

See Also Document object, Page object, Shape object

Shape Object

Shape Object

The **Shape** object is used for controlling and obtaining information about a shape in a ConceptDraw document. Provides access to virtually all elements and characteristics of a shape. In ConceptDraw Basic, the **Shape** object describes all ConceptDraw shapes except service objects - that is, 1D-shapes, simple 2D-shapes, groups, connectors, smart connectors, vector pictures, object that contain raster images, and other. Depending on the shape type to which the instance of the **Shape** object corresponds, the object supports different methods and properties.

Properties

Name	Supported Types	Description
Angle	All	The angle to which the given object is rotated relative to the coordinate system of the parent shape.
BeginX	1D-shape	The X coordinate of the begin point of the shape.
BeginY	1D-shape	The Y coordinate of the begin point of the shape.

ConnectObjBegin	1D-shape, connectors, smart-connectors	The **ID** of the shape, to which the begin point of this 1D-shape is connected.
ConnectObjEnd	1D-shape, connectors, smart-connectors	The **ID** of the shape, to which the end point of this 1D-shape is connected.
ConnectTypeBegin	1D-shape, connectors, smart-connectors	The connection type of the connector's begin point to the shape.
ConnectTypeEnd	1D-shape, connectors, smart-connectors	The connection type of the connector's begin point to the shape.
DblClick	All	The shape's double-click action.
DblClickAction	All	The user-defined double-click action.
Desc	All	The description of the shape.
Document	All	Read-only. Returns the document, to which the shape belongs.
EndX	1D-shape	The X-coordinate of the end point of the shape.
EndY	1D-shape	The Y-coordinate of the end point of the shape.
FillColor	1D-shape, 2D-shape	Read-only. The fill color of the shape.
FillPatColor	1D-shape, 2D-shape	Read-only. The fill pattern color of the shape.
FillPattern	1D-shape, 2D-shape	The type of the fill pattern of the shape.
FlipX	All	A flag that specifies whether the shape is flipped horizontally.
FlipY	All	A flag that specifies whether the shape is flipped vertically.
FlowAroundObjects	Smart-connector	A flag that specifies whether the smart connector should flow around other shapes, located on the same page as this smart connector (**True** - flow around, **False** - pass through).
GPinX	All	The X-coordinate of the rotation center of the shape in the coordinate system of the parent group.
GPinY	All	The X-coordinate of the rotation center of the shape in the coordinate system of the parent group.
Height	All	The height of the shape.
Hyperlink	All	The **ID** of the hyperlink, assigned to the shape.
ID	All	Read-only. ID of the shape.
Is1D	All	Read-only. **True** if 1D-shape, **False** if 2D-shape or other object type.
Layer	All	The ID of the layer to which the shape belongs.

LineBegin	1D-shape	The begin arrowhead type of a 1D-shape.
LineEnd	1D-shape	The end arrowhead type of a 1D-shape.
LineEndSize	1D-shape	The size of begin and end arrowheads of a 1D-shape.
LockAspect	All	A flag that protects the shape from unproportional resizing.
LockBegin	1D-shape	A flag that protects the begin point of a 1D-shape from repositioning with the mouse.
LockCalcWH	All	A flag that specifies whether to update the alignment box size if the coordinates of the shape's vertices have been changed.
LockConnector	Smart-connector	A flag that doesn't allow the smart connector to re-route automatically.
LockDelete	All	A flag that protects the shape from deleting.
LockEnd	1D-shape	A flag that protects the end point of a 1D-shape from repositioning with the mouse.
LockFlipX	All	A flag that protects the shape from flipping horizontally.
LockFlipY	All	A flag that protects the shape from flipping vertically.
LockHeight	All	A flag that protects the shape's height when the shape is resized.
LockMoveX	All	A flag that protects the shape from horizontal repositioning.
LockMoveY	All	A flag that protects the shape from vertical repositioning.
LockRotate	All	A flag that protects the shape from rotation.
LockTextBound	1D-shape, 2D-shape	A flag that protects the shape on the border of the text object to go beyond the boundaries of the object.
LockVertex	All	A flag that protects the vertices from modifying with the mouse.
LockWidth	All	A flag that protects the shape's width when the shape is resized.
LPinX	All	The X offset of the shape's rotation center (GPin) with respect to the center of the shape's coordinate system.
LPinY	All	The Y offset of the shape's rotation center (GPin) with respect to the center of the shape's coordinate system.

Name	All	The shape's name.
NonPrinting	All	A flag that specifies whether to print the shape when the document is printed: **True** - don't print shape, **False** - print shape.
ObjType	All	The shape type: simple shape, group, vector picture, etc.
Page	All	The page to which the shape belongs.
Parent	All	The parent group (the group to which this shape belongs).
PenColor	1D-shape, 2D-shape	Read-only. The line color for this shape.
PenPattern	1D-shape, 2D-shape	The line pattern for this shape.
PenWeight	1D-shape, 2D-shape	The line width for this shape.
RoundCorners	1D-shape, 2D-shape	The corner radius of the shape.
ShadowColor	1D-shape, 2D-shape	The shape's shadow color.
ShadowPatColor	1D-shape, 2D-shape	The shape's shadow pattern color.
ShadowPattern	1D-shape, 2D-shape	The shape's shadow pattern type.
ShowAlignBox	All	A flag that sets whether to display the shape's alignment box.
ShowControlHandles	All	A flag that sets whether to display the shape's control handles.
ShowShapeHandles	All	A flag that sets whether to display the shape's resize and rotation handles.
ShowText	All	A flag that sets whether to display the shape's text.
SubID	All	Read-only. The unique number of the shape within its parent object (group or page).
Text	All	The string that contains the text of the shape.
TextAngle	All	The angle to which the object text is rotated with respect to the coordinate system of the shape.
TextBlock	All	The shape's text block.
TextFlipX	All	A flag that specifies whether the shape's text is flipped horizontally. **True** - text is flipped, **False** - text is not flipped.
TextFlipY	All	A flag that specifies whether the shape's text is flipped vertically. **True** - text is flipped, **False** - text is not flipped.
TextGPinX	All	The X-coordinate of the rotation center of the shape's text block.

TextGPinY	All	The Y-coordinate of the rotation center of the shape's text block.
TextHeight	All	The text block height.
TextLPinX	All	The X offset of the rotation center of the shape's text block with respect to the center of the shape's coordinate system.
TextLPinY	All	The Y offset of the rotation center of the shape's text block with respect to the center of the shape's coordinate system.
TextWidth	All	The text block width.
Width	All	The shape's height.

Methods

Name	Supported Types	Description
Action	All	Returns a user-defined action by its index in the user-defined action collection of the shape.
ActionsNum	All	Returns the number of user-defined actions of the shape.
AddAction	All	Adds a new user-defined action.
AddConnectDot	All	Adds a new connection point to the shape.
AddControlDot	All	Adds a new control handle to the shape.
AddCustomProp	All	Adds a new connection point custom property to the shape.
AddDataSource	All	Adds a new data source to the collection of data source object (shape).
AddDSValue	all	Adds a new row containing the field Value, in the Data Table parameters of the object (shape).
AddGeometry	1D-shape, 2D-shape	Adds a new geometry to the geometry collection of the shape.
AddVariable	All	Adds a new user variable to the variable collection of the shape.
ArcTo	All	Draws an arc.
BeginShape	group	Creates a shape in the group which is later considered as the current Basic shape of the group, or returns the current Basic shape of the group.
Character	All	Returns a character block by its index in the character block collection of the shape.

CharactersNum	All	Returns the number of character blocks in the shape.
ColorProperty	All	Returns the color of the specified shape property. The property is defined by the constant tag and the indexes of the geometry and segment of the shape to which it belongs.
ConnectDot	All	Returns a connection point by its index in the connection point collection of the shape.
ConnectDotsNum	All	Returns the number of connection points in the connection point collection of the shape.
ControlDot	All	Returns a control handle by its index in the control handle collection of the shape.
ControlDotsNum	All	Returns the number of control handles in the control handle collection of the shape.
ConvertToGroup	group	Converts a ConceptDraw Vector Picture object to a ConceptDraw group preserving its position in the document.
ConvertToVFPicture	group	Converts a ConceptDraw shape to a Vector Picture object preserving its position in the document.
CSVColorValue	all	Returns an instance of **Color,** which contains information about the color, the value of which are located at the specified position in the table view a CSV file of the specified data source object (shape).
CSVGetColumnForKey	all	Returns the number of columns found by searching on a key in a table view of this CSV file data source object (shape).
CSVMinRowLength	all	Returns the minimum number of lines (from all the rows) in a tabular representation of a CSV file of the specified data source object (shape).
CSVRowLength	all	Returns the number of elements in the specified row in a table view of this CSV file data source object (shape).
CSVRowMaxElement	all	Returns the maximum element of the specified row in a table view of this CSV file data source object (shape).
CSVRowMinElement	all	Returns the minimum element of the specified row in a table view of this CSV file data source object (shape).

CSVRowNum	all	Returns the number of rows in a table view of this CSV file data source object (shape).
CSVText	all	Returns the text that are in the specified position in the table view a CSV file of the specified data source object (shape).
CSVTextForKey	all	Returns the text found by searching on a key in a table view of this CSV file data source object (shape).
CSVValue	all	Returns an integer value that is at the specified position in the table view of this CSV file data source object (shape).
CSVValueD	all	Gets a value that is in the specified position in the table view of this CSV file data source object (shape).
CSVValueDForKey	all	Returns the value found using the search key in a table view of this CSV file data source object (shape).
CSVValueForKey	all	Returns the integer value found by searching on a key in a table view of this CSV file data source object (shape).
CSVValueType	all	Returns the type of data that resides in the specified position in the table view a CSV file of the specified data source object (shape).
CustomProp	All	Returns a custom property by its index in the custom property collection of the shape.
CustomPropByLabel	All	Returns a custom property by its label in the custom property collection of the shape.
CustomPropsNum	All	Returns the number of custom properties of the shape.
DataSource	All	Returns a collection of data from the data source object (shape) of the index.
DataSourcesNum	All	Returns the number of data sources in the collection of the object (shape).
DrawConnector	group	Draws a connector.
DrawGroup	group	Creates a group inside the given group.
DrawGuide	group	Draws a guide line.
DrawLine	All	Draws a line.
DrawOval	All	Draws an ellipse.
DrawRect	All	Draws a rectangle.
DrawSector	All	Draws a sector of the circle.

DrawSmartConnector	group	Draws a smart connector.
DrawStamp	group	Draws inside the group a copy of the specified shape with the specified size and position, same as the Stamp tool in ConceptDraw.
DrawStampSelection	group	Draws inside the group copies of all selected shapes with the specified size and position, same as the Stamp tool in ConceptDraw.
DropStamp	group	Creates inside the group a copy of the specified shape, same as the Stamp tool in ConceptDraw, preserving the size of the original shape.
DropStampSelection	group	Creates inside the group copies of all selected shapes, same as the Stamp tool in ConceptDraw, preserving the size of the original shapes.
DSValue	all	Returns an instance of an object DataSourceValue, containing data from a table row Data parameters of the object (shape) of the index.
DSValueEl	all	Returns an instance of an object by name DataSourceValue line (field Name) Data Table parameters of the object (shape), containing in the Value data list.
DSValuesNum	all	Returns the number of rows in a table Data parameters of the object (shape).
EndShape	group	Returns the current Basic shape of the group and informs ConceptDraw that creation of the shape is finished.
Equal	All	Copies all the properties and the contents of the specified shape to the given shape.
ExcelColorValue	All	Returns an instance of **Color,** which contains information about the color, the value of which are located at the specified position in the table view XLS file specified data source object.
ExcelGetColumnForKey	All	Returns the column number, found by searching on a key in a table view XLS file specified data source object.
ExcelMinRowLength	All	Returns the minimum number of lines (from all the rows) in the table view XLS file specified data source object.
ExcelRowLength	All	Returns the number of elements in the specified row in a table view XLS file specified data source object.

ExcelRowMaxElement	All	Returns the maximum element of the row in the table view XLS file specified data source object.
ExcelRowMinElement	All	Returns the minimum element of the row in the table view XLS file specified data source object.
ExcelRowNum	All	Returns the number of rows in a table view XLS file specified data source object.
ExcelText	All	Returns the text written in a specified position in the table view XLS file specified data source object.
ExcelTextForKey	All	Returns the text found by searching on a key in a table view XLS file specified data source object (shape).
ExcelValue	All	Returns an integer value that is at the specified position in the table view XLS file specified data source object (shape).
ExcelValueD	All	Gets a value that is at the specified position in the table view XLS file specified data source object.
ExcelValueDForKey	All	Returns the value found using the search key in a table view XLS file specified data source object.
ExcelValueForKey	All	Returns the integer value found by searching on a key in a table view XLS file specified data source object.
ExcelValueType	All	Returns the type of data that resides in the specified position in the table view XLS file specified data source object.
FileText	All	Returns the text written in that text file data source object.
GeometriesNum	1D-shape, 2D-shape	Returns the number of geometries in the shape.
Geometry	1D-shape, 2D-shape	Returns a geometry by its index in the geometry collection of the shape.
GetBooleanProperty	All	Returns the value of a **Boolean** type property.
GetByteProperty	All	Returns the value of a **Byte** type property.
GetCharacterIndex	All	Returns the index of the character block which includes the character with the specified index in the line of the shape's text.
GetDoubleProperty	All	Returns the value of a **Double** type property.

GetIndex	All	Returns the index of the object (shape) in a collection of objects (shapes) of the parent group.
GetIntegerProperty	All	Returns the value of an **Integer** type property.
GetLongProperty	All	Returns the value of a **Long** type property.
GetParagraphIndex	All	Returns the index of the paragraph in the shape's paragraph collection by the specified character's index in the shape's text.
GetPropertyFormula	All	Returns the property's formula in the form of a string.
GetShapeByName	group	Searches for the object (shape) with the given name of the group.Returns the position of the found object (shape) in a collection of objects (shapes) of the group.
GetSingleProperty	All	Returns the value of a **Single** type property.
GetStringProperty	All	Returns the value of a **String** type property
GPtoLp	All	Performs the conversion of the coordinates of the coordinate system of the parent object (shape) (group or page) in the local coordinate system of (this) object (shape).
InsertPicture	group	Inserts into a group an object that contains picture from the specified file.
IsDefaultFormula	All	Returns **True** if the specified property of the shape has a formula that is marked as default. Otherwise returns **False.**
IsNullFormula	All	Returns **True** if the specified property of the shape has no formula.Otherwise returns **False.**
LAtoWA	All	Converts the specified angle from local coordinates of this shape into global coordinates.
LineTo	1D-shape, 2D-shape, group	Draws a line in the shape.
LPtoGP	All	Converts the coordinates of the point from local coordinates of this shape into the coordinate system of the parent shape (group or page).
LPtoWP	All	Converts the coordinates of the point from local coordinate system of this shape into global coordinates.
MoveTo	1D-shape, 2D-shape, group	Sets the position of the current point of the shape, used for drawing shapes.

Paragraph	All	Returns a paragraph by its index in the paragraph collection of the shape.
ParagraphsNum	All	Returns the number of paragraphs in the shape's text.
PropertyChanged	All	Tells the ConceptDraw engine that the value of the specified table property has been changed and the formulas of the dependent properties must be re-calculated.
RecalcProperty	All	Tells the ConceptDraw engine that the value of the specified table property is to be re-calculated using its table formula.
RemoveAction	All	Removes a user-defined action by its index in the user-defined action collection of the shape.
RemoveAllServObjs	group	Removes all service objects that belong to the group.
RemoveAllShapes	group	Removes all shapes that belong to the group.
RemoveCharacter	All	Removes a character block by its index in the character block collection of the shape.
RemoveConnectDot	All	Removes a connection point by its index in the connection point collection of the shape.
RemoveControlDot	All	Removes a control handle by its index in the control handle collection of the shape.
RemoveCustomProp	All	Removes a user-defined action by its index in the user-defined action collection of the shape.
RemoveDataSource	All	Deletes the data source from the collection of data sources, the object (shape) of the index.
RemoveDSValue	All	Removes a row from a table Data parameters of the object (shape) of the index.
RemoveGeometry	1D-shape, 2D-shape	Removes a geometry by its index in the geometry collection of the shape.
RemoveParagraph	All	Removes a paragraph by its index in the paragraph collection of the shape.
RemoveServObj	group	Removes a service object by its index in the service object collection of the group.
RemoveServObjByID	group	Removes a service object with the specified **ID** (the ID property) from the service object collection of the group.
RemoveShape	group	Removes a shape by its index in the shape collection of the group.

RemoveShapeByID	group	Removes a shape with the specified ID from the shape collection of the group.
RemoveVariable	All	Removes a user-defined variable by its index in the user-defined variable collection of the shape.
ReorderServObj	group	Moves the service object to the specified position in the service object collection of the group.
ReorderServObjByID	group	Moves a service object with the specified ID to the indicated position in the service object collection of the group.
ReorderShape	group	Moves the shape to the specified position in the shape collection of the group.
ReorderShapeByID	group	Moves a shape with the specified ID to the indicated position in the shape collection of the group.
SendBack	all	Moves the object (shape) in the first position in the collection of objects (shapes) of the parent group.
SendFront	all	Moves the object (shape) in the last position in the collection of objects (shapes) of the parent group.
ServObj	group	Returns a service object by its index in the service object collection of the group.
ServObjByID	group	Searches for a service object with the specified ID in the service object collection of the group.
ServObjsNum	group	Returns the number of service objects in the group.
SetBooleanProperty	All	Sets the value of a **Boolean** type property by the specified tag.
SetByteProperty	All	Sets the value of a **Byte** type property by the specified tag.
SetCharColor	All	Sets color for the specified character block of the shape's text.
SetCharFont	All	Sets font for the specified character block of the shape's text.
SetCharHyperlink	All	Sets hyperlink for the specified character block of the shape's text.
SetCharLanguage	All	Sets encoding for the specified character block of the shape's text.

SetCharPos	All	Sets position (subscript, superscript) for the specified character block of the shape's text.
SetCharSize	All	Sets font size for the specified character block of the shape's text.
SetCharSpacing	All	Sets character spacing for the specified character block of the shape's text.
SetCharStyle	All	Sets font style (bold, italic, underline, etc.) For the specified character block of the shape's text.
SetDefaultFormula	All	Creates a default formula for the specified table property of the shape.
SetDoubleProperty	All	Sets a value of the specified **Double** type table property.
SetIntegerProperty	All	Sets a value of the specified **Integer** type table property.
SetLongProperty	All	Sets a value of the specified **Long** type table property.
SetNullFormula	All	Removes the formula from the specified table property of the shape.
SetParaAfterSpacing	All	Sets spacing between the specified and next paragraph of the shape's text.
SetParaBeforeSpacing	All	Sets spacing between the specified and previous paragraph of the shape's text.
SetParaFirstInd	All	Sets the first line indent for the specified paragraph of the shape.
SetParaHAlign	All	Sets horizontal alignment type for the specified paragraph relative to the text box.
SetParaLeftInd	All	Sets the distance to the left edge of the text box for the specified paragraph of the shape.
SetParaLineSpacing	All	Sets the line spacing for the specified paragraph of the shape.
SetParaRightInd	All	Sets the distance to the right edge of the text box for the specified paragraph of the shape.
SetPropertyFormula	All	Sets a table formula for the specified table property of the shape.
SetSingleProperty	All	Sets a value of the specified **Single** type table property.
SetStringProperty	All	Sets a value of the specified **String** type table property.
SetStyle	All	Assigns a style with the specified name to the shape.

Shape	group	Returns a shape by its index in the shape collection of the group.
ShapeByID	group	Searches for a shape with the specified ID in the shape collection of the group.
ShapeBySubID	group	Searches for a shape with the specified SubID in the shape collection of the group.
ShapesNum	group	Returns the number of shapes in the group.
SplineStart	1D-shape, 2D-shape, group	Starts drawing a new spline.
SplineTo	1D-shape, 2D-shape, group	Creates a spline segment in the shape.
StepBack	all	Moves the object (shape) back by one position in the collection of objects (shapes) of the parent group.
StepFront	all	Moves the object (shape) by one position in the collection of objects (shapes) of the parent group.
Variable	All	Returns a user-defined variable by its index in the user-defined variable collection of the shape.
VariablesNum	All	Returns the number of user-defined variables contained in the shape.
WPtoLP	All	Converts the coordinates of the specified point from the global coordinate system to the local coordinate system of this shape.
XPathText	All	Returns the text written in the specified XML file data source object.
XPathValue	All	Returns the integer value from the specified XML file data source object.
XPathValueD	All	Returns the value of the specified XML file data source object.

Remarks

A ConceptDraw shape can be located on a document page, inside a group of shapes, or be stored inside a library object (**Master** object) - that is, inside a library. Which page or group contains its own collection of ConceptDraw shapes, and uses the appropriate methods of the **Page** and **Shape** objects to control them. Each library object can contain only one ConceptDraw shape. An instance of the Shape object can also be retrieved by using properties of other objects (**Shape**, **ServObj**, **Window**) which refer to the shape to which they belong.

The **Shape** object has some properties, known as table properties of the shape - that is, the properties which can be associated with a table formula. To work with such properties as with table properties the appropriate methods of the **Shape** object are used (see above). Note, that if a table property was modified, one should use the **RecalcProperty** and **PropertyChanged** methods to re-calculate the depending properties and re-draw the shape respectively.

Also the methods of the Shape object provide control over connection points of connectors, control handles, user-defined actions, variables, custom properties, text block parameters, paragraphs and character blocks of the shape's text, shape geometries, etc.

An instance of the Shape object can be retrieved by using the following methods and properties:
Master object: Shape property.
Page and **Shape** object: Shape method, ShapeByID method and other
ServObj and **Shape** object: Parent property.
Window object: Shape property.

Also an instance of the **Shape** object can be retrieved by using the **thisShape** global variable, pre-defined at the shape script level. **thisShape** returns the shape which script is being executed when the variable is referred to.

See Also	Character object, Color object, Document object, Hyperlink object, Master object, Page object, Paragraph object, ServObj object, TextBlock object, Window object

Style Object

Style Object

The **Style** object describes the ConceptDraw document style. It allows to control the following style properties: line color, fill color, line weight, various text parameters, etc.

Properties

Character	Read-only. Text display parameters for this style.
EndsSize	The end arrows size - 1D-shape for this style.
FillColor	Read-only. The fill color.
FillPatColor	Read-only. The pattern fill color.
FillPattern	The pattern fill type.

HasCharAttr	A flag that specifies whether the **Character** property is effective.
HasEndsAttr	A flag that specifies whether the arrowhead properties for 1D shapes are effective: **LineBegin**, **LineEnd** and **LineEndSize**.
HasFillAttr	A flag that specifies whether the shape's fill properties are effective in this style: **FillColor**, **FillPatColor** and **FillPattern**.
HasParaAttr	A flag that specifies whether the **Paragraph** property is effective.
HasPenAttr	A flag that specifies whether the shape's line properties are effective in this style: **PenColor**, **PenPattern** and **PenWeight**.
HasShadowAttr	A flag that specifies whether the shape's shadow properties are effective in this style: **ShadowColor**, **ShadowPatColor** and **ShadowPattern**.
HasTxtblockAttr	A flag that specifies whether the **TextBlock** property is effective.
LineBegin	The begin arrowhead type of a 1D-shape.
LineEnd	The end arrowhead type of a 1D-shape.
LineEndSize	The size of begin and end arrowheads of a 1D-shape.
Name	Read-only. The name of the style. The unique name that defines the style within the scope of the style collection of the document.
Paragraph	Read-only. Paragraph parameters.
PenColor	Read-only. The color of the shape's lines.
PenPattern	The shape's line pattern.
PenWeight	The shape's line weight.
ShadowColor	Read-only. The shape's shadow color.
ShadowPatColor	Read-only. The shape's shadow pattern color.
ShadowPattern	The shape's shadow pattern type.
TextBlock	Read-only. The text block parameters.

Methods

SetFillColor	Sets the fill color (pattern) of an object (shape) for the current style of the document.
SetFillPatColor	Sets the color of the fill pattern of the object (shape) for the current style of the document.
SetPenColor	Sets the line color of the object (shape) for the current style of the document.
SetShadowColor	Sets the color of the shadow of the object (shape) for the current style of the document.

SetShadowPatColor	Sets the color of the pattern (pattern), the shadow of the object (shape) for the current style of the document.

Remarks

Each ConceptDraw document has its own style collection. The **Document** object has methods for working with style collections. However, styles are intended for describing properties of ConceptDraw shapes. So, it's possible to apply style properties to a ConceptDraw shape by using the SetStyle method of the **Shape** object. To set the default style for new shapes, created in the document, use the **DefStyle** property of the **Document** object.

An instance of the **Style** object can be retrieved by using the following methods of the **Document** object: AddStyle method, Style method, StyleByName method.

Alternatively, the style properties can be changed from within ConceptDraw, menu "Format->Define Styles".

See Also	Character object, Color object, Document object, Paragraph object, TextBlock object, Shape object

TabStop Object

TabStop Object

The **TabStop** object controls tab stop properties: text alignment relative to the tab stop, tab stop position.

Properties

Align	Determines the alignment of the text with respect to the tab stop.
Pos	The distance between the tab stop position and the left edge of the text block, where this tab stop is located.

Remarks

Tab stop positions of a text block specify positions to which the insertion point jumps when you hit **Tab** when editing text. Each **TextBlock** object contains its own tab stop collection and the corresponding methods for working with that collection.

An instance of the **TabStop** object can be retrieved by using the following methods of the **TextBlock** object: AddTabStop method, TabStop method.

See Also Shape object, TextBlock object

TextBlock Object

TextBlock Object

The **TextBlock** object provides access to various text block properties of a ConceptDraw shape, such as the dimensions of the text block, background color, tab stop properties and more.

Properties

VAlign	Vertical alignment type of the text within the text block.
TopMargin	The top margin of the text block.
BottomMargin	The bottom margin of the text block.
LeftMargin	The left margin of the text block.
RightMargin	The right margin of the text block.
TextBkgnd	Read-only. Text block background color.
DefTabStop	The default tab stop distance from the left edge of the text block.

Methods

AddTabStop	Adds a new tab stop to the tab stop collection of the text block.
RemoveTabStop	Removes the specified tab stop and returns the number of remaining tab stops.
TabStop	Returns a **TabStop** object corresponding to a tab stop with the specified index in the tab stop collection of the text block.
TabStopsNum	Returns the number of tab stops in a text block.

Remarks

A text block is used to describe text properties of a ConceptDraw shape (TextBlock property (Shape object)). An instance of the **TextBlock** object is also contained in the document

(DefTextBlock property) to describe the default text block settings for new shapes, and in a document style (TextBlock property (Style object)) to describe the text block settings applied to shapes when the style is assigned to them.

See Also Document object, Shape object, Style object, TabStop object

Variable Object

Variable Object

A service object. You may need variables when several different fields use result of the same calculations. So, the additional variables can be used to store the results. You may also use the additional variables to store various object parameters, which you're working with, so that you don't have to refer to them. An instance of this object can be retrieved from the Shape Object.

Properties

X	The X-coordinate of the point.
Y	The Y-coordinate of the point.

See Also Shape object

Window Object

Window Object

The **Window** object is used to get information about and to control windows in ConceptDraw. The following window types exist in ConceptDraw: document view, library view, table view and ConceptDraw Basic Editor window.

Properties

Property	**Supported Window Types**	**Description**
ID	all windows	Read-only. Returns the window ID.
Type	all windows	Read-only. Returns the window type.
State	all windows	Read-only. Returns the state of the window.
Left	all windows	Read-only. Return the X coordinate of the top left point of the window.
Top	all windows	Read-only. Return the Y coordinate of the top left point of the window.
Height	all windows	Read-only. Returns the window height in pixels.
Width	all windows	Read-only. Returns the window width in pixels.
ViewZoom	document view	The zoom level set in this window.
ViewCenterX	document view	Read-only. Returns the X coordinate of the point, displayed in the center of the window.
ViewCenterY	document view	Read-only. Returns the X coordinate of the point, displayed in the center of the window.
Document	document view	Read-only. Returns the document which contents is displayed in the window.
Library	library view	Read-only. Returns the active library in the library window.
Page	document view	Read-only. Returns the document page displayed in the window.
Shape	document view, table view	Read-only. Returns the shape displayed in the window, if the window is the Edit Group window.

Methods

FindLib	library view	Returns the index of the library in the library collection of the window.
GetSelectedService	document view	Returns the index of the library in the library collection of the window.
GetSelectedShape	document view	Returns the index of the library in the library collection of the window.
Lib	library view	Returns a library by its index in the library collection of the window.

LibByName	library view	Returns a library by the specified name (the **Name** property).
LibsNum	library view	Returns the number of the libraries, opened in the window.
Deselect	document view	Deselect a shape by the specified ID (the **ID** property) of the shape.
DeselectAll	document view	Deselects all shapes on the page displayed in the window.
GetSelectedService method	document view	Returns a service object from the collection-selected objects (shapes) is displayed in a window or group of pages to index.
GetSelectedShape method	document view	Returns an object (shape)-selected objects from the collection (shapes) is displayed in a window or group of pages to index.
Select	document view	Selects a shape by the specified ID (the **ID** property) of the shape.
SelectAll	document view	Selects all shapes on the page displayed in the window.
SelectedNum	document view	Returns the number of selected shapes on the page, displayed in the window.
SetWindowRect	document view	Sets the size and position of the window.
Minimize	all windows	Minimizes the window.
Maximize	all windows	Maximizes the window.
Restore	all windows	Restores the original size and position of the window.
ScrollViewTo	document view	Scrolls the window to the point with the specified coordinates.

Remarks

An instance of the **Window** object can be retrieved by using the following methods and properties:
The **Application** object: ActiveLibWnd property, LibWindowByID method, FirstLibWindow method, NextLibWindow method.
The **Document** object: ActiveView property, ViewByID method, FirstView method, NextView method.

See Also Application object, Document object

ConceptDraw access Objects Properties

Action Property (Action object)

Action Property

A Byte value. Gets or sets the result of the formula, assigned to the shape.

Applies to objects: Action

Syntax

[**Let**] *RetVal* = *object*.**Action**

[**Let**] *object*.**Action** = *SetVal*

The **Action** property syntax has these Elements:

Element	**Description**
object	A reference to an instance of the shape.
RetVal	A Byte value, the result of execution of the formula.
SetVal	A Byte value, the result of execution of the formula.

Remarks

To set the formula, use the SetPropertyFormula method of the Shape Object.

Example

This example demonstrates how the Action property can be used.

```
Dim s as Shape, MyAction as Action
' Assume the shape with ID 1 is on the active page.
' Also assume the shape contains at least one action.
s = thisDoc.ActivePage.ShapeByID(1)
' Take reference to an instance of the Action object
Set MyAction = s.Action(1)
' Output the result of the funciton defined in the action.
trace MyAction.Action
```

See Also Action Object, Shape Object, SetPropertyFormula Method, ActionsNum Method, AddAction Method, Action Method, RemoveAction Method

Action Property (DataSource object)

A **String** type property. The action that will be done in the case of new data from the source.

Applies to: DataSource object

Syntax

[[**Let**] `ActionRet` =] *object*.**Action**

[**Let**] *object*.**Action** = `ActionSet`

The **Action** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **DataSource** object.
`ActionRet`	Optional. A **String** type variable.
`ActionSet`	Required. An expression that returns a **String** value.

Remarks

The **Action** property is also a table parameter of the DataSource, that is, its value can be described by a formula. To work with **Action** as a table parameter, use the **CDPT_DS_ACTION** constant tag.

Example

```
dim ds as DATASOURCE
ds = thisShape.DATASOURCE(1)
trace ds.Action
ds.Action = "Time"
trace ds.Action

or

thisShape.SetPropertyFormula("_CALLTHIS(""Function Name"")", CDPT_DS_ACTION,
1)
```

```
trace ds.Action
```

See Also	DataSource object, Active property, DataSource property, Refresh property, ShowErrors property, ShowWarnings property, Timeout property

ActiveDoc Property

ActiveDoc Property

Read-only. Returns the active Document object, the document shown in the active window.

Applies to: Application object

Syntax

[[**Set**] *docRet* =] *object*.**ActiveDoc**

The **ActiveDoc** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an **Application** object.
docRet	Optional. A variable of the **Document** type.

Remarks

Only one document from the documents, open in the application, can be active. When no documents are open, there is no active document and the **ActiveDoc** property returns the value **Nothing**.

Example

This example contains an application-level script. It demonstrates using the **ActiveDoc** property.

```
Dim active_doc as Document              ' Declare variable
set active_doc = thisApp.ActiveDoc      ' Get active document
active_doc.Name = "Current_doc"         ' Seta a new filename to  active
document
' ... here may be some code for inflation of your document
active_doc.Save()                       ' Save active document with a new name
```

```
TRACE active_doc.FullName              ' Display full filename of saved
document
```

See Also [Document object]

ActiveLayer Property

ActiveLayer Property

Long property. Gets or sets the document's active layer ID (the **ID** property). The active layer is the layer to which shapes are assigned when dropped on the drawing page.

Applies to: Document object

Syntax

[[**Let**] *layerIDRet* =] *object*.**ActiveLayer**

[**Let**] *object*.**ActiveLayer** = *layerIDSet*

The **ActiveLayer** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
layerIDRet	Optional. A **Long** variable.
layerIDSet	Required. An expression that returns a **Long** value. The **ID** (**ID** property) of the layer to be set as the active layer of the document.

Remarks

There is always an active layer in the document, because a ConceptDraw document always contains at least one layer. If there is no layer with the ID, specified by *layerIDSet* in the document, the value of the **ActiveLayer** property is not modified. Use the **LayerByID** method to retrieve an instance of the **Layer** object by the layer ID.

Example

This example contains a document-level script. The script draws two rectangles on the first page of the document. The first rectangle has ID **1**, the second rectangle has ID **2**.

```
' Make active the  layer with ID 1
thisDoc.ActiveLayer = 1
' Draw rectangle on the active layer
thisDoc.Page(1).DrawRect(100,100,700,400)
' Make active the  layer with ID 2.
thisDoc.ActiveLayer = 2
' Draw rectangle on the active layer
thisDoc.Page(1).DrawRect(300,300,900,600)
```

See Also	ID property, Layer method, LayerByID method, LayersNum method, Layer object

ActiveLibWnd Property

ActiveLibWnd Property

Read-only. Returns an instance of the **Window** object, corresponding to the active library window.

Applies to: Application object

Syntax

[[**Set**] *libWindowRet* =] *object*.**ActiveLibWnd**

The **ActiveLibWnd** property syntax has these Elements:

Element	**Description**
object	Required. An expression returning an .**Application** object.
libWindowRet	Optional. A **Window** type variable.

Remarks

If **ActiveLibWnd** returns **Nothing**, there is no active library window in the application, and no active library.However, it doesn't mean that there are no open libraries in the application. Note, that an active library window always contains an active library.

Example

This example contains an application-level script. It demonstrates an attempt to find a bug in ConceptDraw Basic by using the fact that the active library window always contains the active library.

```
' Declare variables
Dim lib_wnd as Window
Dim m_lib as Library
' Get active library window
Set lib_wnd = thisApp.ActiveLibWnd
If lib_wnd <> Nothing Then
 ' Get active library in the active library window
 Set m_lib = lib_wnd.Library
 if m_lib <> thisApp.ActiveLib Then
  MsgBox("Oh, NO! You ve found a BUG! Please report this at 
www.conceptdraw.com as soon as you can!")
 else
  MsgBox("Wanna find some bugs? No, our company cannot help you.")
 End If
Else
 MsgBox("There is no active library window now!")
End If
```

See Also Window object, Library object

ActiveLib Property

ActiveLib Property

Read-only. Returns an instance of the **Library** object corresponding to the active library.

Applies to: Application object

Syntax

[[**Set**] *libRet* =] *object*.**ActiveLib**

The **ActiveLib** property syntax has these Elements:

Element	**Description**
object	Required. An expression returning an **Application** object.
libRet	Optional. A **Library** type variable.

Remarks

If there is no active library in the application, the **ActiveLib** property returns **Nothing**. However, it doesn't mean that there are no open libraries in the application. Use the **SetActiveLib** method to make a library active.

Example

This example contains an application-level script. It demonstrates using the **ActiveLib** property.

```
Dim active_lib as Library                ' Declare a  Library type variable
Set active_lib = thisApp.ActiveLib       ' Set active library
active_lib.Name = "Current_Lib"          ' Give new filename to active library
'... some code to inflate your active lib
active_lib.Save()                        ' Save library with new filename.
TRACE active_lib.FullName                ' Display full filename and path to the
saved library
```

See Also SetActiveLib method, Library object

ActivePage Property

ActivePage Property

Read-only. Returns an instance of the **Page** object corresponding to the active page of the document.

Applies to: Document object

Syntax

[[**Set**] *pageRetl* =] *object*.**ActivePage**

The **ActivePage** property syntax has these Elements:

Element	**Description**
object	Required. An expression returning a **Document** object.
pageRet	Optional. A **Page** type variable.

Remarks

Note, that a document always has at least one page, and one page is always active. If there is more than one page in the document, the active page is the one displayed in the active document view.

To set a new active page use the **SetActivePage** method. If you address the **ActivePage** property when the active window is note a page view (for instance, it's the ConceptDraw Basic Editor window), the **ActivePage** property returns the most recent active page.

Example

This example contains a document-level script. It draws rectangles on the first three pages of the document, using the **ActivePage** property to address the pages. The **SetActivePage** is used to set active pages in this order: Page 1, then Page 2, then Page 3.

```
' If there are less than 3 pages in the document
' then add remaining pages
If thisDoc.PagesNum() < 3 Then
    thisDoc.AddPage()
End If
If thisDoc.PagesNum() < 3 Then
    thisDoc.AddPage()
End If
' Set page 1 as active page
thisDoc.SetActivePage(1)
' Draw rectangle on page 1
thisDoc.ActivePage.DrawRect( 100, 100, 700, 500 ).Text = 1
' Set page 2 as active page
thisDoc.SetActivePage(2)
' Draw rectangle on page 2
thisDoc.ActivePage.DrawRect( 100, 100, 700, 500 ).Text = 2
' Set page 3 as active page
thisDoc.SetActivePage(3)
' Draw rectangle on page 3
thisDoc.ActivePage.DrawRect( 100, 100, 700, 500 ).Text = 3
```

See Also SetActivePage method, SetActivePageByID method, Page object

ActiveView Property

ActiveView Property

Read-only. Returns an instance of the **Window** object, corresponding to the active document view.

Applies to: Document object

Syntax

[[**Set**] *windowRet* =] *object*.**ActiveView**

The **ActiveView** property syntax has these Elements:

Element	Description
object	Required. An expression returning a **Document** object.
windowRet	Optional. A **Window** type variable.

Remarks

If there is no active document view, the **ActiveView** property returns a **Window** object, corresponding to the most recent active page of the document (the **ActivePage** property). It doesn't necessarily return the last active document view, rather any window, corresponding to the most recent active page of the document. The active document view type (the **Type** property) is *document view* (cdDOCVIEW).

Example

This example contains a document-level script. It demonstrates how to control the active document view by using the **ActiveView** property of the document.

```
' Maximize window
thisDoc.ActiveView.Maximize()
' Display message
MsgBox( "Active View is Maximized!" )
' Minimize active window
thisDoc.ActiveView.Minimize()
MsgBox( "Active View is Minimized!" )
' Restore the original view of the window
thisDoc.ActiveView.Restore()
MsgBox( "Active View is Restored!" )
```

See Also ActivePage property, Type property, Window object

Active Property

Active Property

A Boolean type property. Provides start or a stop of process of updating of data from a source. True - start of process of updating of data from a source. False - a stop of process of updating of data from a source. By default Active property is equal to False.

Applies to: DataSource object

Syntax

[[**Let**] *ActiveRet* =] *object*.**Active**

[**Let**] *object*.**Active** = *ActiveSet*

The **Active** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **DataSource** object.
ActiveRet	Optional. A **Boolean** type variable.
ActiveSet	Required. An expression that returns a **Boolean** value.

Remarks

The **Active** property is also a table parameter of the DataSource, that is, its value can be described by a formula. To work with **Active** as a table parameter, use the **CDPT_DS_ACTIVE** constant tag.

Example

```
dim ds as DATASOURCE
ds = thisShape.DATASOURCE(1)
trace ds.Active
ds.Active = True
trace ds.Active
or
thisShape.SetPropertyFormula("False",CDPT_DS_ACTIVE, 1)
trace ds.Active
```

See Also DataSource object, Action property, DataSource property, Refresh property, ShowErrors property, ShowWarnings property, Timeout property

Address Property

Address Property

Read-only. A **String** value. Represents a path to the file or URL to which the hyperlink points.

Applies to: Hyperlink object

Syntax

[[**Set**] *addressRet* =] *object*.**Address**

The **Address** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Hyperlink** object.
addressRet	Optional. A **String** type variable.

Remarks

This property is efficient if the hyperlink has the type **cdLinkToFile** or **cdLinkToURL** (the **LinkType** property).

See Also	AddHyperlinkToDocument method, AddHyperlinkToFile method, AddHyperlinkToURL method

AfterSpacing Property

AfterSpacing Property

A **Single** type property. Specifies the amount of space inserted after each paragraph in the shape's text block.

Applies to: Paragraph object

Syntax

[**Let**] *singleRet* = *object*.**AfterSpacing**

[**Let**] *object*.**AfterSpacing** = *afterSpacingSet*

The **AfterSpacing** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Paragraph** object.
singleRet	Optional. A **Single** type variable.
afterSpacingSet	Required. An expression that returns a **Single** value.

Remarks

The interval is specified in **InternalUnit** (internal units of measure of ConceptDraw).

The **AfterSpacing** property is also a table parameter of the shape, to which the *object* paragraph belongs. That is, its value can depend on a formula. To work with **AfterSpacing** as a table parameter, use the **CDPT_PARA_AFTERSPACING** constant tag.

Example

This example shows how to increase the spacing between the second and the third paragraphs. It assumes that there is a shape with at least three paragraphs of text on the current page.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Increase spacing between second and third paragraphs by 150 points
s.Paragraph(2).AfterSpacing = 150
' Inform ConceptDraw Engine about the changes to recalculate and redraw the
document
s.PropertyChanged(CDPT_PARA_AFTERSPACING)
```

See Also SetParaAfterSpacing method

Align Property

Align Property

A **Byte** type property. Specifies the horizontal alignment type of text with respect to the current tab stop.

Applies to: TabStop object

Syntax

[[**Let**] *byteRet* =] *object*.**Align**

[**Let**] *object*.**Align** = *alignSet*

The **Align** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the **TabStop** object.
byteRet	Optional. A **Byte** type variable.

alignSet	Required. An expression that returns a **Byte** type value.

Remarks

The following constants show the possible alignment values:

Constant	**Value**	**Description**
cdTabStopLeft	0	Alignment to the left edge.
cdTabStopCenter	1	Alignment by the center.
cdTabStopRight	2	Alignment to the right edge.
cdTabStopDecimal	3	Alignment by the decimal point (for point-delimited real numbers).
cdTabStopComma	4	Alignment by the decimal comma (for comma-delimited real numbers).

The **Align** property is also a table parameter of the shape, which contains the text block with the *object* tab stop inside. That is, its value can depend on a formula. To work with **Align** as a table parameter, use the **CDPT_TABALIGN** constant tag.

Example

This example demonstrates using the **Align** property. It assumes, that the active page contains the shape with ID1, which has text. Also, at least one tab stop is defined.

```
Dim s as Shape, MyTabStop as TabStop
' Assume shape with  ID 1 exists on page.
' Assume the shape's text contains several numbers
' located on different lines and having point as decimal separator
s = thisDoc.ActivePage.ShapeByID(1)
' Add tab stop that will obtain number 1
s.TextBlock.AddTabStop()
' Get reference to a TabStop object
Set MyTabStop = s.TextBlock.TabStop(1)
' Set properties for the first tab stop.
' Numbers in different lines must be aligned by their decimal points.
MyTabStop.Pos = 200
MyTabStop.Align = cdbTabStopDecimal
' Inform ConceptDraw Engine about the changes to recalculate and redraw the
document
s.PropertyChanged(CDPT_TABALIGN)
s.PropertyChanged(CDPT_TABPOS)
```

See Also Pos property, Shape object, TextBlock object

Angle Property

Angle Property

A **Double** type property. Represents the angle to which the shape is rotated clockwise around its rotation center (**GPinX**, **GPinY** properties). The angle is measured with respect to the horizontal axis, in the coordinate system of the parent shape (parent group or page).

Applies to: ServObj object, Shape object

Syntax

[[**Let**] *angleRet* =] *object*.**Angle**

[**Let**] *object*.**Angle** = *angleSet*

The **Angle** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Window** object.
heightRet	Optional. A **Double** type variable.
angleSet	Required. An expression that returns a **Double** value.

Remarks

The **Angle** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **Angle** as a table parameter, use the **CDPT_ANGLE** constant tag.

The angle values are specified in radians.

See Also	Angle property, GPinX property, GPinY property, FlipX property, FlipY property, Height property, LPinX property, LPinY property, Width property

Author Property

Author Property

String property. Gets or sets a string containing the name of the author of the document/library.

Applies to: Document object, Library object

Syntax

[[**Let**] *authorRet* =] *object*.**Author**

[**Let**] *object*.**Author** = *authorSet*

The **Author** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object from the **Applies to** list.
authorRet	Optional. A **String** type variable.
authorSet	Required. An expression that returns a **String** value. The string to be set as the name of the author of the document.

Remarks

For a new document the **Author** property contains an empty string.

Setting the **Author** property for a document is equivalent to entering information in the Author box in the Document Properties dialog box, tab General (click Document Properties on the File menu). To set **Author** for a library, use the Properties dialog from the File ->Library menu.

Example

The example below contains a document-level script. It demonstrate how to view the author information for a document by creating a **Text** object with the "DocAuthor" formula.

```
Dim shp As Shape                                    ' Declare variables
Set shp = thisDoc.ActivePage.DrawRect(100,100,1000,300) ' Draw rectangle
shp.Text = ""
shp.SetPropertyFormula( "DocAuthor", CDPT_TEXT )      ' Set formula for the
Text property
shp.RecalcProperty( CDPT_TEXT )                     ' Re-calculating the Text
property
thisDoc.Author = "New Document Author"              ' Set new value for the
Author property
```

See Also Company property, Desc property, Subj property, Title property

BackPageID Property

BackPageID Property

A **Long** type property. Gets or sets the ID (**ID** property) of the background page for the specified page.

Applies to: Page object

Syntax

[[**Let**] *longRet* =] *object*.**BackPageID**

[**Let**] *object*.**BackPageID** = *backPageIDSet*

The **BackPageID** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Page** object.
longRet	Optional. A **Long** type variable.
backPageIDSet	Required. An expression that returns a **Long** value.

Remarks

The contents of a background page is displayed on the background of the page it's assigned to, but can't be edited. The *backPageIDSet* returns an ID of a page within the same document (the **Document** property) which can be set as background page (the **IsBackGround** property). A background page can be set as background for itself, that is the **BackPageID** property is only effective, if the **IsBackground** value is **False** for the page.

See Also ID property, Document property

BeforeSpacing Property

BeforeSpacing Property

A **Single** type property. The amount of space inserted before each paragraph in the shape's text block.

Applies to: Paragraph object

Syntax

[**Let**] *singleRet* = *object*.**BeforeSpacing**

[**Let**] *object*.**BeforeSpacing** = *beforeSpacingSet*

The **BeforeSpacing** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Paragraph** object.
singleRet	Optional. A **Single** type variable.
beforeSpacingSet	Required. An expression that returns a **Single** value.

Remarks

The interval is specified in **InternalUnit** (internal units of measure of ConceptDraw).

The **BeforeSpacing** property is also a table parameter of the shape, to which the *object* paragraph belongs. That is, its value can be defined by a formula. To work with **BeforeSpacing** as a table parameter, use the **CDPT_PARA_BEFORESPACING** constant tag.

Example

This example shows how to increase the spacing between the first and the second paragraphs. It assumes that there is a shape with at least three paragraphs of text on the current page.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Increase spacing between the first and second paragraphs by 150 points
s.Paragraph(2).BeforeSpacing = 150
' Inform ConceptDraw Engine about the changes to recalculate and redraw the
document
s.PropertyChanged(CDPT_PARA_BEFORESPACING)
```

See Also

SetParaBeforeSpacing method

BeginX Property

BeginX Property

A **Double** type property. The X-coordinate of the begin point of the 1D shape.

Applies to: Shape object

Syntax

[[**Let**] *beginXRet* =] *object*.**BeginX**

[**Let**] *object*.**BeginX** = *beginXSet*

The **BeginX** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
beginXRet	Optional. A **Double** type variable.
beginXSet	Required. An expression that returns a **Double** value.

Remarks

This property is effective for 1D-shapes only. To determine the shape type, use the following properties: **Is1D**, **ObjType**.

The **BeginX** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **BeginX** as a table parameter, use the **CDPT_BEGINX** constant tag.

The unit of measure for the coordinates are is **InternalUnit**.

See Also	BeginY property, EndX property, EndY property, Is1D property, ObjType property

BeginY Property

BeginY Property

A **Double** type property. The Y-coordinate of the begin point of the 1D shape.

Applies to: Shape object

Syntax

[[**Let**] *beginYRet* =] *object*.**BeginY**

[**Let**] *object*.**BeginY** = *beginYSet*

The **BeginY** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
beginYRet	Optional. A **Double** type variable.
beginYSet	Required. An expression that returns a **Double** value.

Remarks

This property is effective for 1D-shapes only. To determine the shape type, use the following properties: **Is1D**, **ObjType**.

The **BeginY** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **BeginY** as a table parameter, use the **CDPT_BEGINY** constant tag.

The unit of measure for the coordinates are is **InternalUnit**.

See Also BeginX property, EndX property, EndY property, Is1D property, ObjType property

Black Property

Black Property

Gets or sets an **Integer** value, that represents the black component of CMYK color.

Applies to: Color object, ColorEntry object

Syntax

[[**Let**] *blackRet* =] *object*.**Black**

[**Let**] *object*.**Black** = *blackSet*

The **Black** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an object from the **Applies to** list.
blackRet	Optional. An **Integer** value.
blackSet	Required. An expression that returns an **Integer** value.

Remarks

The **Black** property is only effective if the color is a CMYK color (see the **IsCMYK** property).

Example

This example contains a document-level script. It demonstrates how to find out the value of the black component of the fill color (in CMYK format) of a Shape object.

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
If s.FillColor.IsCMYK<> false  Then    ' A CMYK color?
 MsgBox(s.FillColor.Black)    ' If yes, display the value of the black
component.
endif
```

See Also Cyan property, Magenta property, Yellow property, IsCMYK property

Blue Property

Blue Property

Gets or sets an **Integer** value, that represents the blue component of an RGB color.

Applies to: Color object, ColorEntry object

Syntax

[[**Let**] *blueRet* =] *object*.**Blue**

[**Let**] *object*.**Blue** = *blueSet*

The **Blue** property syntax has these Elements:

Element	Description

object	Required. An expression that returns an object from the **Applies to** list.
blueRet	Optional. An **Integer** value.
blueSet	Required. An expression that returns an **Integer** value.

Remarks

The **Blue** property is only effective if the color is a RGB color (see the **IsRGB** property).

Example

This example contains a document-level script. It demonstrates how to find out the value of the blue component of the fill color (in RGB format) of a Shape object.

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
If s.FillColor.IsRGB <> false  Then    ' An RGB color ?
 MsgBox(s.FillColor.Blue)    ' If yes, display the value of the blue component
endif
```

See Also Blue property, Green property, Red property, IsRGB Property

BottomMargin Property

BottomMargin Property

A **Single** type property. Specifies the distance between the bottom border of the text box and the last line of text it contains.

Applies to: TextBlock object

Syntax

[[**Let**] *singleRet* =] *object*.**BottomMargin**

[**Let**] *object*.**BottomMargin** = *bottomMarginSet*

The **BottomMargin** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **TextBlock** object.

singleRet	Optional. A **Single** type variable.
bottomMargin Set	Required. An expression that returns a **Single** value.

Remarks

The unit of measure for the **BottomMargin** property is **InternalUnit**.

The **BottomMargin** property is also a table parameter of the shape, to which the *object* text block belongs - that is, its value can be described by a formula. To work with **BottomMargin** as a table parameter, use the **CDPT_BOTTOMMARGIN** constant tag.

Example

This example shows how to increase the distance between the bottom border of the text box and the last line of text it contains. It assumes the shape exists and contains text.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Increase the distance between the bottom border of the text box and the last
line of text by 20 points.
s.TextBlock.BottomMargin = 20
' Inform ConceptDraw Engine about the changes to recalculate and redraw the
document
s.PropertyChanged(CDPT_BOTTOMMARGIN)
```

Bottom Property

Bottom Property

Gets or sets a Double value, representing the coordinates of the bottom point of an instance of the shape.

Applies to objects: DRect

Syntax

[**Let**] *RetVal* = *object*.**Bottom**

[**Let**] *object*.**Bottom** = *SetVal*

Element	**Description**
object	A reference to an instance of the shape.

RetVal	A Double type variable
SetVal	A Double value

Example

```
Dim MyObject as new DRect   ' Create an instance of the shape
MyObject.Bottom = 200
```

See Also

DRect Object

Caption Property

Caption Property

A **String** value. Gets or sets the name of a menu or menu item.

Applies to: Menu object, MenuItem object

Syntax

[[**Let**] *captionRet* =] *object*.**Caption**

[**Let**] *object*.**Caption** = *captionSet*

The **Caption** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object from the **Applies to** list.
captionRet	Optional. A **String** value.
captionSet	Required. An expression that returns a **String** value.

Remarks

The Caption property contains the name of the menu or a menu item as it's displayed in ConceptDraw for the upper-level user-defined menu, obtained with the **CustomMenu** property.

Example

This example contains a document-level script.

```
dim mainMenu as Menu, myMenuItem as MenuItem
' Get reference to a Menu object from thisDoc
mainMenu = thisDoc.CustomMenu
' Remove all exisitng menu items
mainMenu.RemoveAll()
' Give a name to mainMenu
mainMenu.Caption = "My Caption"
' Add a MenuItem object to mainMenu
myMenuItem = mainMenu.AddMenuItem(0)
' Name it myMenuItem
myMenuItem.Caption = "My Caption 2"
```

Character Property

Character Property

Read-only. Returns a **Character** object that contains the character block parameters for this style.

Applies to: Style object

Syntax

[[**Let**] *characterRet* =] *object*.**Character**

The **Character** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Style** object.
characterRet	Optional. A **Character** type variable.

Remarks

You can't modify the **Character** object, stored in the **Character** property of the style, however you can change the attributes of this instance of the **Character** object. When a style is assigned to a shape, the parameters of the **Character** property of the style are applied to all character blocks of the shape. The **Character** property is only effective if the **HasCharAttr** property of this style is **True**.

See Also	HasCharAttr property, Paragraph property, TextBlock property, Character object

Checked Property

Checked Property

A **Boolean** value. Gets and sets the state of a check mark beside the command name on the menu.

Applies to: Action object, MenuItem object

Syntax

[[**Let**] *checkedRet* =] *object*.**Checked**

[**Let**] *object*.**Checked** = *checkedSet*

The **Checked** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **MenuItem** object.
checkedRet	Optional. A **Boolean** type variable.
checkedSet	Required. An expression that returns a **Boolean** value.

Remarks

If **True**, displays a check mark beside the command name on the menu. If **False**, the check mark is not displayed.

Example

This example contains a document-level script.

```
Dim s as Shape, MyAction as Action
' Assume the shape with ID 1 exists on the active page.
' Assume the shape contains at least one action.
s = thisDoc.ActivePage.ShapeByID(1)
' Get a reference to the  Action object
Set MyAction = s.Action(1)
' Set check mark
MyAction.Checked = True
```

CmdID Property

CmdID Property

Read-only. A **Long** value. Identifier of a menu or a menu item.

Applies to: Menu object, MenuItem object

Syntax

[[**Let**] *cmdIDRet* =] *object*.**CmdID**

The **CmdID** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **MenuItem** object.
cmdIDRet	Optional. A **Long** type variable.

Remarks

This identifier represents a unique integer number, associated with a user-defined item or a menu item within the ConceptDraw application.

Colored Property

Colored Property

A Boolean value, that indicates whether a layer is colored.

Applies to objects: Layer

Syntax

[[**Let**] *RetVal* =] *object*.**Colored**

[**Let**] *object*.**Colored** = *SetVal*

The **Colored** property syntax has these Elements:

Element	Description
object	A reference to an instance of the object.
RetVal	A Boolean type variable.
SetVal	A Boolean value.

Remarks

If the **Colored** property is TRUE, all objects on the layer are displayed in the color, defined by the Color property. Otherwise the objects are displayed in their original colors.

Example

This example demonstrates using the **Colored** property.

```
Dim MyLayer as Layer
' Get Layer 2 from thisDoc
set MyLayer = thisDoc.Layer(2)
' Make it colored
MyLayer.Colored = True
```

See Also Layer Object, Document Object

Color Property

Color Property

Read-only. Returns a **Color** object that corresponds to the color of an instance of the object from the **Applies To** list.

Applies to: Character object, Layer object

Syntax

[[**Set**] *colorRet* =] *object*.**Color**

The **Color** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object from the **Applies to** list.
colorRet	Optional. A **Color** type variable.

Remarks

When *object* is a block of characters, its color means the color of all characters in this character block. The **Color** property is also a table parameter of the shape to which this character block belongs, that is, its value can be described by a formula. To work with **Color** as a table parameter, use the **CDPT_CHAR_COLOR** constant tag.

When *object* is a layer, its color means the color of all shapes that belong to that layer, providing the layer is colored (the **Colored** property).

See Also Colored property, Count property, SetCharColor method, Color object

Comment Property

Comment Property

Gets or sets a String value, that represents a comment or prompt.

Applies to objects: ControlDot

Syntax

[**Let**] *RetVal* = *object*.**Comment**

[**Let**] *object*.**Comment** = *SetVal*

The **Comment** property syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
RetVal	A String type variable.
SetVal	A String value.

Example

This example demonstrates using the **Comment** property.

```
Dim MyControlDot as ControlDot, MyShape As Shape
MyShape = thisDoc.ActivePage.DrawRect(50,50,500,500)    ' Create a Shape
object
MyControlDot = MyShape.AddControlDot()
MyControlDot.X = 100 ' Set coordinates for control handle
MyControlDot.Y = 150
MyControlDot.Comment = "Wise Dot"   ' Set prompt
' Inform ConceptDraw engine about changes
MyShape.PropertyChanged(CDPT_CONTROL_X)
MyShape.PropertyChanged(CDPT_CONTROL_Y)
MyShape.PropertyChanged(CDPT_CONTROL_COMMENT)
```

Company Property

Company Property

String property. Returns or sets the value of the Company field in a document's or library's properties.

Applies to: Document object, Library object

Syntax

[[**Let**] *companyRet* =] *object*.**Company**

object.**Company** = *companySet*

The **Company** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object from the **Applies to** list.
companyRet	Optional. A **String** type variable.
companySet	Required. An expression that returns a **String** value.

Remarks

For a new document or library the **Company** property returns an empty string. Setting the **Company** property for a document is equivalent to entering information in the Company field in the Document Properties dialog box, tab General (click Document Properties on the File menu). To set **Company** for a library, use the Properties dialog from the File ->Library menu.

Example

This example contains an application-level script. The program asks the user from a company name, an assigns this name to the **Company** property of all open documents, which don't have this property set.

```
' Declare variables
Dim cur_doc As Document
Dim str_company As String
' Ask user to enter company name
str_company = InputBox("Enter company name:")
```

```
' If user input nothing, then quit program
If str_company = "" Then
 MsgBox("You did not enter anything!")
 End
End If
' Check all open documents starting from the end of the list
For i=thisApp.DocsNum() To 1 Step -1
 ' Get document
 Set cur_doc = thisApp.Doc(i)
 ' If no company name is set, set company name provided by user
 If cur_doc.Company = "" Then
  cur_doc.Company = str_company
 End If
Next i
```

See Also Title property, Author property, Subj property, Desc property

ConnectObjBegin Property

ConnectObjBegin Property

A **Long** type property. ID (the **ID** property) of the shape, to which the begin point of this 1D-shape is connected.

Applies to: Shape object

Syntax

[[**Let**] *shapeIDBeginRet* =] *object*.**ConnectObjBegin**

[**Let**] *object*.**ConnectObjBegin** = *shapeIDBeginSet*

The **ConnectObjBegin** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
shapeIDBeginRet	Optional. A **Long** type variable.
shapeIDBeginSet	Required. An expression that returns a **Long** value. The ID (ID property) of the shape, to which the begin point of this shape is to be connected.

Remarks

This property is only effective for 1D-shapes. If the shape with *shapeIDBeginSet* ID is not found in the shape collection of the page to which *object* belongs, the **ConnectObjBegin** property doesn't change its value. Also notice that an 1D-shape can't be connected to itself.

If **ConnectObjBegin** is modified, *object* is completely rebuilt and connected to the new shape, defined by *shapeIDBeginSet.*

See Also	ConnectObjEnd property, ConnectTypeBegin property, ConnectTypeEnd property

ConnectObjEnd Property

ConnectObjEnd Property

A **Long** type property. ID (the **ID** property) of the shape, to which the end point of this 1D-shape is connected.

Applies to: Shape object

Syntax

[[**Let**] *shapeIDEndRet* =] *object*.**ConnectObjEnd**

[**Let**] *object*.**ConnectObjEnd** = *shapeIDEndSet*

The **ConnectObjEnd** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
shapeIDEndRet	Optional. A **Long** type variable.
shapeIDEndSet	Required. An expression that returns a **Long** value. The ID (ID property) of the shape, to which the end point of this shape is to be connected.

Remarks

This property is only effective for 1D-shapes. If the shape with *shapeIDEndSet* ID is not found in the shape collection of the page to which *object* belongs, the **ConnectObjEnd** property doesn't change its value. Also notice that an 1D-shape can't be connected to itself.

If **ConnectObjEnd** is modified, *object* is completely rebuilt and connected to the new shape, defined by *shapeIDEndSet.*

See Also	ConnectObjBegin property, ConnectTypeBegin property, ConnectTypeEnd property

ConnectTypeBegin Property

ConnectTypeBegin Property

A **Byte** type property. Determines the connection type of the connector's begin point to the shape.

Applies to: Shape object

Syntax

[[**Let**] *typeBeginRet* =] *object*.**ConnectTypeBegin**

[**Let**] *object*.**ConnectTypeBegin** = *typeBeginSet*

The **ConnectTypeBegin** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
typeBeginRet	Optional. A **Byte** type variable.
typeBeginSet	Required. An expression that returns a **Byte** value.

Remarks

Below are the possible values of **ConnectTypeBegin**:

Constant	**Value**	**Description**
CDCT_NoConnect	0	The endpoint of connector is not connected to any other shape.
CDCT_Left	1	Connected to the middle of the shape's left side.
CDCT_Top	2	Connected to the middle of the shape's top side.
CDCT_Right	3	Connected to the middle of the shape's right side.
CDCT_Bottom	4	Connected to the middle of the shape's bottom side.
CDCT_Centre	5	Connected to the side of the shape's alignment box that is nearest to the other end of the connector.

CDCT_ConnectDot	255	Connected to a connection point.

When **ConnectTypeBegin** is modified, *object* is rebuilt and re-connected to the shape with the new connection type.

See Also ConnectObjBegin property, ConnectObjEnd property, ConnectTypeEnd property

ConnectTypeEnd Property

ConnectTypeEnd Property

A **Byte** type property. Determines the connection type of the connector's end point to the shape.

Applies to: Shape object

Syntax

[[**Let**] *typeEndRet* =] *object*.**ConnectTypeEnd**

[**Let**] *object*.**ConnectTypeEnd** = *typeEndSet*

The **ConnectTypeEnd** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
typeEndRet	Optional. A **Byte** type variable.
typeEndSet	Required. An expression that returns a **Byte** value.

Remarks

Below are the possible values of **ConnectTypeEnd**:

Constant	**Value**	**Description**
CDCT_NoConnect	0	The endpoint of connector is not connected to any other shape.
CDCT_Left	1	Connected to the middle of the shape's left side.
CDCT_Top	2	Connected to the middle of the shape's top side.

CDCT_Right	3	Connected to the middle of the shape's right side.
CDCT_Bottom	4	Connected to the middle of the shape's bottom side.
CDCT_Centre	5	Connected to the side of the shape's alignment box that is nearest to the other end of the connector.
CDCT_ConnectDot	255	Connected to a connection point.

When **ConnectTypeEnd** is modified, *object* is rebuilt and re-connected to the shape with the new connection type.

See Also ConnectObjBegin property, ConnectObjEnd property, ConnectTypeBegin property

Count Property

Count Property

Read-only. A **Long** type property. Indicates the number of characters in the character block or paragraph.

Applies to: Character object, Paragraph object

Syntax

[[**Let**] *countRet* =] *object*.**Count**

The **Count** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
countRet	A **Long** type property.

Remarks

Always returns a value equal to or greater than **1** as there's always at least one character in a character block/paragraph.

See Also

CustomMenu Property

CustomMenu Property

Read-only. Returns an instance of the **Menu** object, corresponding to the user-defined menu of the application/document.

Applies to: Application object, Document object

Syntax

[[**Set**] *menuRet* =] *object*.**CustomMenu**

The **CustomMenu** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the object from the **Applies to** list.
menuRet	Optional. A **Menu** type variable.

Remarks

There are two types of user-defined menu: of the application and of the document (the **Application** and **Document** objects respectively). The user-defined menu is located in the Tools menu of ConceptDraw and is visible if contains at least one item. The Tools menu can incorporate no more than two user-defined menus at a time - one for the application, the other for the active document (the **ActiveDoc** property).

Example

This example contains an application-level script. It displays the properties of the application's user-defined menu **CustomMenu**.

```
thisApp.CustomMenu.Caption = "App Custom Menu"
thisApp.CustomMenu.Prompt = "App Custom Menu Prompt"
TRACE "-------------------------"
TRACE "thisApp.CustomMenu.CmdID   = " & thisApp.CustomMenu.CmdID
TRACE "thisApp.CustomMenu.Caption = " & thisApp.CustomMenu.Caption
TRACE "thisApp.CustomMenu.Prompt  = " & thisApp.CustomMenu.Prompt
TRACE "thisApp.CustomMenu.Enabled = " & thisApp.CustomMenu.Enabled
TRACE "------------------------"
```

See Also Menu object

Cyan Property

Cyan Property

Gets or sets an **Integer** value, that represents the cyan component of CMYK color.

Applies to: Color object, ColorEntry object

Syntax

[[**Let**] *cyanRet* =] *object*.**Cyan**

[**Let**] *object*.**Cyan** = *cyanSet*

The **Blue** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an object from the **Applies to** list.
cyanRet	Optional. An **Integer** value.
cyanSet	Required. An expression that returns an **Integer** value.

Remarks

The **Cyan** property is only effective if the color is a CMYK color (see the **IsCMYK** property).

Example

This example contains a document-level script. It demonstrates how to find out the value of the cyan component of the fill color (in CMYK format) of a Shape object.

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
If s.FillColor.IsCMYK<> false  Then    ' A CMYK color?
 MsgBox(s.FillColor.Cyan)    ' If yes, display the value of the cyan
component.
endif
```

See Also Magenta property, Yellow property, Black property, IsCMYK property

DataSource Property

DataSource Property

A String type property. A relative or full way to a source of data.

Applies to: DataSource object

Syntax

[[**Let**] *dataSourceRet* =] *object*.**DataSource**

[**Let**] *object*.**DataSource** = *dataSourceSet*

The **DataSource** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **DataSource** object.
dataSourceRet	Optional. A **String** type variable.
dataSourceSet	Required. An expression that returns a **String** value.

Remarks

The **DataSource** property is also a table parameter of the DataSource, that is, its value can be described by a formula. To work with **DataSource** as a table parameter, use the **CDPT_DS_DATASOURCE** constant tag.

Example

```
dim ds as DATASOURCE
ds = thisShape.DATASOURCE(1)
trace ds.DataSource
ds.DataSource = "TxtSource.txt"
trace ds.DataSource
or
thisShape.SetPropertyFormula("""TxtSource.txt""",CDPT_DS_DATASOURCE, 1)
trace ds.DataSource
```

See Also DataSource object, Action property, Active property, Refresh property, ShowErrors property, ShowWarnings property, Timeout property

DblClickAction Property

DblClickAction Property

An **Integer** type property. Determines the user-defined action when the shape is double-clicked. The user-defined action will be effective only if the **DblClick** property has the same value as DBLCLICK_ACTION.

Applies to: Shape object

Syntax

[[**Let**] *dblClickActionRet* =] *object*.**DblClickAction**

[**Let**] *object*.**DblClickAction** = *dblClickActionSet*

The **DblClickAction** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dblClickActionRet	Optional. A **Byte** type variable.
dblClickActionSet	Required. An expression that returns a **Byte** value. Represents the index of the user-defined action in the user-defined actions of the shape. The valid range is from 0 to 256.

Remarks

The **DblClickAction** property specified the index in the collection of the user-defined action to be performed when the shape is double-clicked. To determine the number of user-defined actions of the shape, use the **ActionsNum** method. If **DblClickAction** equals **0**, the next action in the collection will be performed each time the shape is double-clicked.

See Also DblClick property, DblClickAction property, Action method, ActionsNum method, Action object

DblClick Property

DblClick Property

A **Byte** type property. Gets or sets the double-click action of the shape.

Applies to: Shape object

Syntax

[[**Let**] *dblClickRet* =] *object*.**DblClick**

[**Let**] *object*.**DblClick** = *dblClickSet*

The **DblClick** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dblClickRet	Optional. A **Byte** type variable.
dblClickSet	Required. An expression that returns a **Byte** value.

Remarks

The **DblClick** property can take the following possible values:

Constant	**Value**	**Description**
DBLCLICK_NOACTION	0	Take no action.
DBLCLICK_EDITTEXT	1	Edit shape's text.
DBLCLICK_OPENGROUP	2	Edit group.
DBLCLICK_SHAPESHEET	3	Open the shape's parameter table.
DBLCLICK_GOTOHLINK	4	Go to the hyperlink.
DBLCLICK_OLE	5	Launch the OLE-application (if the shape is embedded).
DBLCLICK_ACTION	9	Perform a user-defined action (the **DblClickAction** property).

See Also DblClick property, DblClickAction property

DefCharacter Property

DefCharacter Property

Read-only. Returns an instance of the **Character** object that corresponds to a sequence of characters, set by default for the text of a shape when it's created in the document.

Applies to: Document object

Syntax

[[**Set**] *characterRet* =] *object*.**DefCharacter**

The **DefCharacter** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Document** object.
characterRet	Optional. A **Character** type variable.

Remarks

The **DefCharacter** property describes a sequence of characters which properties are assigned by default to the text of the new shapes created in the document. Note, that newly created shapes don't contain text, and therefore don't contain any instances of the **Character** object. That's why the property will be applied to the shape at the moment the text is assigned to it, provided the shape didn't contain any text before. That is, when text is added to the shape, a sequence of characters fully identical to the **DefCharacter** property and including all the text assigned to the shape, is added to the character sequence collection of the shape.

To set default parameters for the paragraphs and text blocks for every new shape in the document, use the **DefParagraph** and **DefTextBlock** properties respectively.

See Also DefParagraph property, DefTextBlock property, Character object, Shape object

DefFillColor Property

DefFillColor Property

Read-only. Returns an instance of the **Color** object that corresponds to the fill color, applied by default to every new shape created in the document.

Applies to: Document object

Syntax

[[**Set**] *colorRet* =] *object*.**DefFillColor**

The **DefFillColor** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
colorRet	Optional. A **Color** type variable.

Remarks

When a new shape is created in the document, the value of the properties of the **Color** object which contains the **DefFillColor** property is set to the corresponding properties of the **Color** object which contains the **FillColor** property of the new shape.

See Also	DefFillPatColor property, DefFillPattern property, FillColor property, Color object, Shape object

DefFillPatColor Property

DefFillPatColor Property

Read-only. Returns an instance of the **Color** object, corresponding to the color of the fill pattern of the shape, set by default to every new shape in the document.

Applies to: Document object

Syntax
[[**Set**] *colorRet* =] *object*.**DefFillPatColor**

The **DefFillPatColor** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Document** object.
colorRet	Optional. A **Color** type variable.

Remarks

When a new shape is created in the document, the value of the properties of the **Color** object which contains the **DefFillPatColor** property is set to the corresponding properties of the **Color** object which contains the **FillPatColor** property of the new shape.

See Also	DefFillColor property, DefFillPattern property, FillPatColor property, Color object, Shape object

DefFillPattern Property

DefFillPattern Property

A **Long** type property. Gets and sets the type of the fill pattern, applied by default to every new shape, created in the document.

Applies to: Document object

Syntax
[[**Let**] *fillPatternRet* =] *object*.**DefFillPattern**

[**Let**] *object*.**DefFillPattern** = *fillPatternSet*

The **DefFillPattern** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Document** object.

fillPatternRet	Optional. A **Long** type variable.
fillPatternSet	Required. An expression that returns a **Long** value.

Remarks

When a new shape is created in the document, the value of the **DefFillPattern** property is set to the corresponding **FillPattern** property of the shape. The range of valid values for the **DefFillPattern** property is the same as for the **FillPattern** property of the **Shape** object.

See Also	DefFillColor property, DefFillPatColor property, FillPattern property, Color object, Shape object

DefParagraph Property

DefParagraph Property

Read-only. Returns an instance of the **Paragraph** object that corresponds to the paragraph, assigned by default to the text of every new shape created in the document.

Applies to: Document object

Syntax

[[**Set**] *paragraphRet* =] *object*.**DefParagraph**

The **DefParagraph** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
paragraphRet	Optional. A **Paragraph** type variable.

Remarks

The **DefParagraph** property contains parameters describing a paragraph, which are applied by default to the text of new shapes created in the document. Note, that new shapes don't contain text, that is, they don't contain any instances of the **Paragraph** object, which describes the paragraph parameters. So, this property will be applied to the shape at the moment the text is assigned to the shape, provided the shape didn't contain any text before. That is, when text is added to the shape, a paragraph, fully identical to the **DefParagraph** paragraph and containing all text assigned to the shape is added to the paragraph collection of the shape.

To set default parameters to the sequence of characters and text block for the new shapes created in the document, use the **DefCharacter** and **DefTextBlock** properties respectively.

See Also DefCharacter property, DefTextBlock property, Paragraph object, Shape object

DefPenColor Property

DefPenColor Property

Read-only. Returns an instance of the **Color** object, that contains information about the line color set by default to every new shape created in the document.

Applies to: Document object

Syntax

[[**Set**] *colorRet* =] *object*.**DefPenColor**

The **DefPenColor** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
colorRet	Optional. A **Color** type variable.

Remarks

When a new shape is created in the document, the value of the properties of the **Color** object which contains the **DefPenColor** property is set to the corresponding properties of the **Color** object which contains the **PenColor** property of the new shape.

See Also DefPenPattern property, DefPenWeight property, PenColor property, Color object, Shape object

DefPenPattern Property

DefPenPattern Property

A **Long** type property. Gets and sets the line pattern applied by default to new shapes created in the document.

Applies to: Document object

Syntax

[[**Let**] *patternRet* =] *object*.**DefPenPattern**

[**Let**] *object*.**DefPenPattern** = *patternSet*

The **DefPenPattern** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the **Document** object.
patternRet	Optional. A **Long** type variable.
patternSet	Required. An expression that returns a **Long** value.

Remarks

When a new shape is created in the document, the value of the **DefPenPattern** property is set to the corresponding **PenPattern** property of the shape. The range of valid values for **DefPenPattern** is the same as for the **PenPattern** property of the **Shape** object.

See Also DefPenColor property, DefPenWeight property, PenPattern property, Shape object

DefPenWeight Property

DefPenWeight Property

A **Long** value. Gets or sets the line weight, set by default to new shapes created in the document.

Applies to: Document object

Syntax

[[**Let**] *weightRet* =] *object*.**DefPenWeight**

[**Let**] *object*.**DefPenWeight** = *weightSet*

The **DefPenWeight** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
weightRet	Optional. A **Long** type variable.
weightSet	Required. An expression that returns a **Long** value.

Remarks

When a new shape is created in the document, the value of the **DefPenWeight** property is set to the corresponding **PenWeight** property of the shape. The range of valid values for **DefPenWeitgh** is the same as for the **PenWeight** property of the **Shape** object.

See Also DefPenColor property, DefPenPattern property, PenWeight property, Shape object

DefShadowColor Property

DefShadowColor Property

Read-only. Returns an instance of the **Color** object that contains information about the shadow color of the shape, which is applied by default to new shapes created in the document.

Applies to: Document object

Syntax

[[**Set**] *colorRet* =] *object*.**DefShadowColor**

The **DefShadowColor** property syntax has these Elements:

Element	Description
object	Required. An expression, that returns a **Document** object.
colorRet	Optional. A **Color** type variable.

Remarks

When a new shape is created in the document the value of the properties of the **Color** object which contains the **DefShadowColor** property is set to the corresponding properties of the **Color** object which contains the **ShadowColor** property of the new shape.

See Also DefShadowPatColor property, DefShadowPattern property, ShadowColor property, Color object, Shape object

DefShadowPatColor Property

DefShadowPatColor Property

Read-only. Returns an instance of the **Color** object, that contains information about the shadow pattern color of the shape, set by default to every new shape in the document.

Applies to: Document object

Syntax

[[**Set**] *colorRet* =] *object*.**DefShadowPatColor**

The **DefShadowPatColor** property syntax has these Elements:

Element	Description
object	Required. An expression, that returns a **Document** object.
colorRet	Optional. A **Color** type variable.

Remarks

When a new shape is created in the document, the value of the properties of the **Color** object which contains the **DefShadowPatColor** property is set to the corresponding properties of the **Color** object which contains the **ShadowPatColor** property of the new shape.

See Also DefShadowColor property, DefShadowPattern property, ShadowPatColor property, Color object, Shape object

DefShadowPattern Property

DefShadowPattern Property

A **Long** type property. Gets and sets the type of the shadow pattern, applied by default to new shapes, created in the document.

Applies to: Document object

Syntax

[[**Let**] *patternRet* =] *object*.**DefShadowPattern**

[**Let**] object.**DefShadowPattern** = *patternSet*

The **DefShadowPattern** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns a **Document** object.
patternRet	Optional. A **Long** type variable.
patternSet	Required. An expression that returns a **Long** value.

Remarks

When a new shape is created in the document, the value of the **DefShadowPattern** property is set to the corresponding **ShadowPattern** property of the shape. The range of valid values for the **DefShadowPattern** property is the same as for the **ShadowPattern** property of the **Shape** object.

See Also DefShadowColor property, DefShadowPatColor property, ShadowPattern property, Shape object

DefStyle Property

A **Long** type property. Gets or sets the style index (the number of the style in the style collection of the document), which is applied by default to new shapes created in the document.

Applies to: Document object

Syntax

[[**Let**] *styleIndexRet* =] *object*.**DefStyle**

[**Let**] *object*.**DefStyle** = *styleIndexSet*

The **DefStyle** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns a **Document** object.
styleIndexRet	Optional. A **Long** type variable.
styleIndexSet	Required. An expression that returns a **Long** value. The index of the style in the style collection of the document.

Remarks

If *styleIndexSet* is less than **-1** or greater than the number of the styles in the style collection of the document, the **DefStyle** property doesn't change its value. Also, the **DefStyle** property can take the following values: **0** - normal style, the style set in ConceptDraw by default, **-1** - no style.

To find out the number of styles in the style collection of the document, use the **StylesNum** method. To retrieve a style by its index in the style collection of the document, use the **Style** method.

See Also	FindStyle method, Style method, StyleByName method, StylesNum method, Style object

DefTabStop Property

DefTabStop Property

A **Single** type value. Represents the default tab stop position from the left edge of the text block. It's used as the default value for new tab stops, added to the tab stop collection of the text block.

Applies to: TextBlock object

Syntax

[[**Let**] *singleRet* =] *object*.**DefTabStop**

[**Let**] *object*.**DefTabStop** = *defTabStopSet*

The **DefTabStop** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **TextBlock** object.
singleRet	Optional. A **Single** type variable.
defTabStopSet	Required. An expression that returns a **Single** value.

Remarks

The value of **DefTabStop** is measured in the internal ConceptDraw units (**InternalUnit**).

The **DefTabStop** property is also a table parameter of the shape, which contains the *object* text block, that is, its value can be described by a formula. To work with **DefTabStop** as a table parameter, use the **CDPT_DEFTABSTOP** constant tag.

Example

This example shows how to modify the interval of the default tab stops in a shape that contains a text block.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Sets the default tab stop interval
s.TextBlock.DefTabStop = 200
' Inform ConceptDraw engine about the changes for re-drawing
s.PropertyChanged(CDPT_DEFTABSTOP)
```

See Also Pos property (TabStop object), AddTabStop method

DefTextBlock Property

Read-only. Returns an instance of the **TextBlock** object that contains parameters of the text block, assigned by default to the text of every new shape created in the document.

Applies to: Document object

Syntax

[[**Set**] *textBlockRet* =] *object*.**DefTextBlock**

The **DefTextBlock** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
textBlockRet	Optional. A **TextBlock** type variable.

Remarks

The **DefTextBlock** property contains parameters describing a text block, which are applied by default to the text of new shapes created in the document. Note, that new shapes don't contain text, that is, they don't contain any instances of the **TextBlock** object, which describes the text block parameters. So, this property will be applied to the shape at the moment text is assigned to the shape, provided the shape didn't contain any text before. That is, when text is added to the shape, a text block, fully identical to the **DefTextBlock** text block and containing all text assigned to the shape is added to the paragraph collection of the shape.

To set default parameters to the sequence of characters and paragraph for the new shapes created in the document, use the **DefCharacter** and **DefParagraph** properties respectively.

See Also	DefCharacter property, DefParagraph property, TextBlock object, Shape object

Desc Property

Desc Property

A **String** type property. Gets or sets a descriptive text string for an object from the **Applies to** list.

Applies to: Document object, Library object, ServObj object, Shape object

Syntax

[[**Let**] *descRet* =] *object*.**Desc**

[**Let**] *object*.**Desc** = *descSet*

The **Desc** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an object in the **Applies to** list.
descRet	Optional. A **String** type variable.
descSet	Required. An expression that returns a **String** value. The string that is set as description for the object in the **Applies To**.

Remarks

The **Desc** property contains an empty string for any new document, library, guide line or shape.

The **Desc** property of any of the objects can also be changed in the dialogs in ConceptDraw: "File->Document Properties->General" for a document, "File->Library->Properties" - for a library, "Format->Shape Properties->Information" - for service objects and simple shapes.

Example

This example contains an application-level script. It adds the last revision date (current date) to the **Desc** property of a document.

```
Dim cur_doc As Document         ' Declare variables
' Get the first document from the document collection of the application
Set cur_doc = thisApp.FirstDoc()
' Loop for all documents of the application
While cur_doc <> Null
    ' Write current date to the end of document description
    cur_doc.Desc = cur_doc.Desc & Chr(13) & Chr(10) & "...was updated: " & Now
    ' Get next document
    Set cur_doc = thisApp.NextDoc()
Wend
```

See Also Author property, Company property, Subj property, Title property

Disabled Property

Disabled Property

A Boolean value. Gets or sets the state of a menu item.

Applies to objects: Action

Syntax

[**Let**] *RetVal* = *object*.**Disabled**

[**Let**] *object*.**Disabled** = *SetVal*

The **Disabled** property syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
RetVal	A Boolean type variable.
SetVal	A Boolean value.

Remarks

If TRUE, the menu item is disabled, otherwise enabled.

Example

This example demonstrates using the **Disabled** property.

```
Dim s as Shape, MyAction as Action
' Assume shape with ID 1 exists on the active page.
' Assume the shape contains at least one Action.
s = thisDoc.ActivePage.ShapeByID(1)
' Get reference to an instance of the  Action object.
Set MyAction = s.Action(1)
' Set Disabled state
MyAction.Disabled = True
```

See Also	Action Object, Shape Object, SetPropertyFormula Method, ActionsNum Method, AddAction Method, Action Method, RemoveAction Method

DocumentsPath Property

DocumentsPath Property

Read-only. A String value. Returns the full way to files which are on the way, adjusted in Preferences appendix dialogue in the Paths tab in the field of Documents.

Applies to: Application object

Syntax

[[**Let**] *DocumentsPathRet* =] *object*.**DocumentsPath**

The **DocumentsPaht** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Application** object.
DocumentsPath Ret	Optional. A **String** type variable.

Remarks

The **DocumentsPath** property by default matters: "**root ConceptDraw DIAGRAM/Samples folder**".

See Also	Application object, HelpPath property, LibrariesPath property, TemplatesPath property

Document Property (Page, ServObj, Shape objects)

Document Property (Page, ServObj, Shape objects)

Read-only. Gets the **Document** object that is associated with the document that contains an object from the **Applies to** list.

Applies to: Page object, ServObj object, Shape object

Syntax

[[**Set**] *documentRet* =] *object*.**Document**

The **Document** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
documentRet	Optional. A **Document** type variable.

Remarks

For shapes stored in a library this property always returns **Nothing**.

See Also

Page property, Parent property, Document object

Document Property

Document Property

Read-only. Gets the **Document** object that is associated with the document whose contents is displayed in this window of ConceptDraw.

Applies to: Window object

Syntax

[[**Set**] *documentRet* =] *object*.**Document**

The **Document** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
documentRet	Optional. A **Document** type variable.

Remarks

This method is only effective if the window is a document window (see the **Type** property). For all other window types the **Document** property always returns **Nothing**.

See Also Library property, Page property, Shape property, Document object

Action Property (DataSource object)

Action Property (DataSource object)

A **String** type property. The action that will be done in the case of new data from the source.

Applies to: DataSource object

Syntax

[[**Let**] *ActionRet* =] *object*.**Action**

[**Let**] *object*.**Action** = *ActionSet*

The **Action** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **DataSource** object.
ActionRet	Optional. A **String** type variable.
ActionSet	Required. An expression that returns a **String** value.

Remarks

The **Action** property is also a table parameter of the DataSource, that is, its value can be described by a formula. To work with **Action** as a table parameter, use the **CDPT_DS_ACTION** constant tag.

Example

```
dim ds as DATASOURCE
ds = thisShape.DATASOURCE(1)
trace ds.Action
ds.Action = "Time"
```

```
trace ds.Action
or
thisShape.SetPropertyFormula("_CALLTHIS(""Function Name"")", CDPT_DS_ACTION,
1)
trace ds.Action
```

See Also DataSource object, Active property, DataSource property, Refresh property, ShowErrors property, ShowWarnings property, Timeout property

Enabled Property

Enabled Property

A **Long** value. A flag that specifies whether a menu or a menu item is enabled or disabled.

Applies to: Menu object, MenuItem object

Syntax

[[**Let**] *enabledRet* =] *object*.**Enabled**

[**Let**] *object*.**Enabled** = *enabledSet*

The **Enabled** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
enabledRet	Optional. A **Boolean** type variable.
enabledSet	Required. An expression that returns a **Boolean** value.

Remarks

If **Enabled** is **TRUE**, the menu or the menu item is operating (enabled). Otherwise either the menu item is disabled, or all items of the menu are disabled (if applied to the menu).

EndsSize Property

EndsSize Property

A **Long** type property. Specifies the ending arrows size 1D-shape.

Applies to: Style object

Syntax

[[**Let**] *longRet* =] *object*.**EndsSize**

[**Let**] *object*.**EndsSize** = *endsSizeSet*

The **EndsSize** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
longRet	Optional. A **Long** type variable.
endsSizeSet	Required. An expression that returns a **Long** value.

Remarks

The value of **LineEnd** can be in the range of **0** to **4**.

Style object:
When a style is assigned to a shape, the parameters of the **endsSize** property of the style are set to the **endsSize** property of the shape.

See Also

LineBegin property, LineEnd property

EndX Property

EndX Property

A **Double** type property. The X-coordinate of the end point of the shape.

Applies to: Shape object

Syntax

[[**Let**] *endXRet* =] *object*.**EndX**

[**Let**] *object*.**EndX** = *endXSet*

The **EndX** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
endXRet	Optional. A **Double** type variable.
endXSet	Required. An expression that returns a **Double** value.

Remarks

This property is only effective for 1D-shapes. The type of a shape can be determined by using the following properties: **Is1D**, **ObjType**.

The **EndX** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **EndX** as a table parameter, use the **CDPT_ENDX** constant tag.

The unit of measure for the specified coordinates is the internal ConceptDraw unit (**InternalUnit**).

See Also	BeginX property, BeginY property, EndY property, Is1D property, ObjType property

EndY Property

EndY Property

A **Double** type property. The X-coordinate of the end point of the shape.

Applies to: Shape object

Syntax

[[**Let**] *endYRet* =] *object*.**EndY**

[**Let**] *object*.**EndY** = *endYSet*

The **EndY** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
endYRet	Optional. A **Double** type variable.
endYSet	Required. An expression that returns a **Double** value.

Remarks

This property is only effective for 1D-shapes. The type of a shape can be determined by using the following properties: **Is1D**, **ObjType**.

The **EndY** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **EndY** as a table parameter, use the **CDPT_ENDY** constant tag.

The unit of measure for the specified coordinates is the internal ConceptDraw unit (**InternalUnit**).

See Also	BeginX property, BeginY property, EndX property, Is1D property, ObjType property

FillColor Property

FillColor Property

Read-only. Returns an instance of the **Color** object that corresponds to the fill color of the shape.

Applies to: Shape object, Style object

Syntax

[[**Set**] *colorRet* =] *object*.**FillColor**

The **FillColor** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
colorRet	Optional. A **Color** type variable.

Remarks

Shape object:
Note, that the shape is filled only when it has closed geometries, that is, geometries whose begin and end points coincide.

The **FillColor** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **FillColor** as a table parameter, use the **CDPT_FILLCOLOR** constant tag.

Style object:
When a style is assigned to a shape, the parameters of the **FillColor** property of the style are set to the **FillColor** property of the shape. **FillColor** is only effective when the **HasFillAttr** property of this style is **True**.

To change the fill pattern color and pattern type of a shape, use the **FillPatColor** and **FillPattern** properties respectively.

See Also	DefFillColor property, FillPatColor property, FillPattern property, HasFillAttr property, Color object

Filled Property

Filled Property

A **Boolean** type property. Gets or sets a flag, that specifies whether to fill or not the area, enclosed by the geometry. If **True**, the geometry is filled, otherwise it isn't filled

Applies to: Geometry object

Syntax

[[**Let**] *filledRet* =] *object*.**Filled**

[**Let**] *object*.**Filled** = *filledSet*

The **Filled** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
filledRet	Optional. A **Boolean** type variable.

filledSet	Required. An expression that returns a **Boolean** value.

Remarks

The **Filled** property is also a table parameter of the shape, that owns the *object* geometry, that is, its value can be described by a formula. To work with **Filled** as a table parameter, use the **CDPT_GEOMETRY_FILLED** constant tag.

See Also Visible property, Shape object

FillPatColor Property

FillPatColor Property

Read-only. Returns an instance of the **Color** object that corresponds to the pattern color of the shape.

Applies to: Shape object, Style object

Syntax

[[**Set**] *colorRet* =] *object*.**FillPatColor**

The **FillPatColor** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
colorRet	Optional. A **Color** type variable.

Remarks

Shape object:
The **FillPatColor** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **FillPatColor** as a table parameter, use the **CDPT_FILLPATCOLOR** constant tag.

Style object:

When a style is assigned to a shape, the parameters of the **FillPatColor** property of the style are set to the **FillPatColor** property of the shape. **FillPatColor** is only effective when the **HasFillAttr** property of this style is **True**.

To change the fill color and pattern type, use the **FillColor** and **FillPattern** properties respectively.

See Also	DefFillPatColor property, FillColor property, FillPattern property, HasFillAttr property, Color object

FillPattern Property

FillPattern Property

A **Long** type property. Gets and sets the fill pattern of the shape.

Applies to: Shape object

Syntax

[[**Let**] *longRet* =] *object*.**FillPattern**

[**Let**] *object*.**FillPattern** = *fillPatternSet*

The **FillPattern** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
longRet	Optional. A **Long** type variable.
fillPatternSet	Required. An expression that returns a **Long** value.

Remarks

The values of the **FillPattern** property can be in the range of **0** to **69**.

Shape object:
The **FillPattern** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **FillPattern** as a table parameter, use the **CDPT_FILLPATTERN** constant tag.

Style object:
When a style is assigned to a shape, the parameters of the **FillPattern** property of the style are set to the **FillPattern** property of the shape. **FillPattern** is only effective when the **HasFillAttr** property of this style is **True**.

To change the fill color and pattern type, use the **FillColor** and **FillPattern** properties respectively.

See Also DefFillPattern property, FillColor property, FillPatColor property, HasFillAttr property, Property Tags Constants

FirstInd Property

FirstInd Property

A **Single** type property. Determines the first line indent for this paragraph.

Applies to: Paragraph object

Syntax

[**Let**] *singleRet* = *object*.**FirstInd**

[**Let**] *object*.**FirstInd** = *firstIndSet*

The **FirstInd** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Paragraph** object.
singleRet	Optional. A **Single** type variable.
firstIndSet	Required. An expression that returns a **Single** value.

Remarks

The distance for the first line indent is specified in the internal ConceptDraw units (**InternalUnit**).

The **FirstInd** property is also a table parameter of the shape that contains the *object* paragraph, that is, its value can be described by a formula. To work with **FirstInd** as a table parameter, use the **CDPT_PARA_FIRSTIND** constant tag.

Example

This example demonstrates how to set a 1 cm indent for the first line of the paragraph in the shape. It assumes that a shape containing text exists on the current page.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Set indent in the first line of the text block's first paragraph.
s.Paragraph(1).FirstInd = 100
' Inform ConceptDraw Engine about changes for re-drawing
s.PropertyChanged(CDPT_PARA_FIRSTIND)
```

See Also SetParaFirstInd method

FlipX Property

FlipX Property

A **Boolean** type property. Specifies whether or not the shape is flipped horizontally. **False** - the shape is not flipped. **True** - the shape is flipped horizontally.

Applies to: Shape object

Syntax

[[**Let**] *flipXRet* =] *object*.**FlipX**

[**Let**] *object*.**FlipX** = *flipXSet*

The **FlipX** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
flipXRet	Optional. A **Boolean** type variable.
flipXSet	Required. An expression that returns a **Boolean** value.

Remarks

The **FlipX** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **FlipX** as a table parameter, use the **CDPT_FLIPX** constant tag.

See Also FilpY property, Property Tags Constants

FlipY Property

FlipY Property

A **Boolean** type property. Specifies whether or not the shape is flipped vertically. **False** - the shape is not flipped. **True** - the shape is flipped vertically.

Applies to: Shape object

Syntax

[[**Let**] *flipYRet* =] *object*.**FlipY**

[**Let**] *object*.**FlipY** = *flipYSet*

The **FlipY** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
flipYRet	Optional. A **Boolean** type variable.
flipYSet	Required. An expression that returns a **Boolean** value.

Remarks

The **FlipY** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **FlipY** as a table parameter, use the **CDPT_FLIPY** constant tag.

See Also FilpX property, Property Tags Constants

FlowAroundObjects Property

FlowAroundObjects Property

A **Boolean** type property.
Shape object: gets or sets the flag that specifies whether the smart connector should avoid shapes on its way, located on the same page as the smart connector (**True** - avoid, **False** - pass through).
Document object: sets the default value for new smart connectors created in the document.

Applies to: Document object, Shape object

Syntax

[[**Let**] *flowAroundRet* =] *object*.**FlowAroundObjects**

[**Let**] *object*.**FlowAroundObjects** = *flowAroundSet*

The **FlowAroundObjects** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
flowAroundRet	Optional. A **Boolean** type variable.
flowAroundSet	Required. An expression that returns a **Boolean** value.

Remarks

If *object* is a document, then when a new smart connector is created, the value of the **FlowAroundObjects** property is set to the **FlowAroundObjects** property of this smart connector. You can also modify the **FlowAroundObjects** property by using the ConceptDraw interface - in the menu "Shape->Connector->Flow Around Objects".

See Also	FlowAroundObjects property, LineJumpOrient property, LineJumpSize property, LineJumpType property, MaxNumberOfLegs property, MinDistToShapes property, PassThroughGroups property

Font Property

Font Property

A **Long** type property. Gets or sets the index of the font in the font collection of the document, used to display the characters of this character block.

Applies to: Character object

Syntax

[[**Let**] *longRet* =] *object*.**Font**

[**Let**] *object*.**Font** = *fontIndexSet*

The **Font** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Character** object.
longRet	Optional. A **Long** type variable.
fontIndexSet	Required. An expression that returns a **Long** value.

Remarks

Each font in the font collection of a ConceptDraw document has has a unique number (index). The fonts are numbered starting from 0. This should be considered when you change the value of the **Font** property.

The **Font** property is also a table parameter of the shape which contains the *object* character block, that is, its value can be described by a formula. To work with **Font** as a table parameter, use the **CDPT_CHAR_FONT** constant tag.

Note

Font numbers may change when fonts are added to or removed from the system. Also keep in mind that font sets are different on different computers. To find out a font's index and the total number of fonts installed on the system, use the FontName, FindFontByName, FontsNum methods of the Document object.

Example

This example is used to change the font of the shape with ID 1, which is located on the current page.

```
Dim MyFontNumber As Long, MyShape As Shape
' Get the index for Times New Roman font
MyFontNumber = thisDoc.FindFontByName("Times New Roman")
' Shape with ID 1 must be on the current page of the document
Set MyShape = thisDoc.ActivePage.ShapeByID(1)
If MyFontNumber <> -1 Then             'if the font is found
'  Set the font (MyShape.Character(1) must exist)
   MyShape.Character(1).Font = MyFontNumber
'  Call PropertyChanged to inform ConceptDraw Engine,
'  that the given property has changed.
```

```
   MyShape.PropertyChanged(CDPT_CHAR_FONT)
EndIf
```

See Also Character object, FontName method , FindFontByName method , FontsNum method

Format Property

Format Property

A String value. Gets or sets the formatting of a custom property.

Applies to objects: CustomProp

Syntax

[[**Let**] *RetVal* =] *object*.**Format**

[**Let**] *object*.**Format** = *SetVal*

The **Format** property syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
RetVal	A String type variable.
SetVal	A String type variable.

Remarks

The **Format** property is only effective when the **Type** property value is a fixed list or a variable list.The **Format** property contains a list of possible values, separated with ";" (semicolon).

Example

This example demonstrates working with the **CustomProp** object.

```
Dim MyShape As Shape, MyProperty as  CustomProp
' Create shape
MyShape = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
'  Create custom properties for MyShape
MyProperty = MyShape.AddCustomProp()
'  Working with the properties of MyProperty
MyProperty.Label = "IP"
MyProperty.Prompt = "TCP/IP address"
MyProperty.Type = 3
MyProperty.Format = "192.168.0.1;192.168.0.2;192.168.0.3"
MyProperty.Value = "192.168.0.1"
MyProperty.Invisible = FALSE
MyProperty.Verify = TRUE
```

See Also CustomProp Object, Document Object

FullName Property

FullName Property

Read-only. A **String** type property. Returns the full name to the document/library file, including the path to the file (the **Path** property) if it has been set, and the name of the file itself (the **Name** property).

Applies to: Document object, Library object

Syntax

[[**Let**] *fullNameRet* =] *object*.**FullName**

The **FullName** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
fullNameRet	Optional. A **String** type variable.

Remarks

The **FullName** property is a combination of the **Path** and **Name** properties of the corresponding objects:

```
thisDoc.FullName = ( thisDoc.Path & thisDoc.Name )                      '
returns True
thisApp.Lib(1).FullName = ( thisApp.Lib(1).Path & thisApp.Lib(1).Name ) '
returns True
```

FullName changes automatically when **Path** or **Name** are changed, and also when the document or library are saved under a new name in ConceptDraw using the Save dialog.

See Also Name property, Path property

GPinX Property

GPinX Property

A **Double** type property. The X-coordinate of the rotaion center of the shape/service object in the coordinate system of the parent group/page.

Applies to: ServObj object, Shape object

Syntax

[[**Let**] *gpinXRet* =] *object*.**GPinX**

[**Let**] *object*.**GPinX** = *gpinXSet*

The **GPinX** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
gpinXRet	Optional. A **Double** type variable.
gpinXSet	Required. An expression that returns a **Double** value.

Remarks

If *object* is an instance of the **Shape** object, the **GPinX** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **GPinX** as a table parameter, use the **CDPT_GPINX** constant tag.

An instance of the parent group (parent object) can be retrieved by using the **Parent** property. To get the page which owns the shape, use the **Page** property.

The unit of measure for the coordinates are the internal ConceptDraw units (**InternalUnit**).

See Also GPingX property, GPinY property, LPinX property, LPinY property, Page property, Parent property, Property Tags Constants

GPinY Property

GPinY Property

A **Double** type property. The Y-coordinate of the rotaion center of the shape/service object in the coordinate system of the parent group/page.

Applies to: ServObj object, Shape object

Syntax

[[**Let**] *gpinYRet* =] *object*.**GPinY**

[**Let**] *object*.**GPinY** = *gpinYSet*

The **GPinY** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
gpinYRet	Optional. A **Double** type variable.
gpinYSet	Required. An expression that returns a **Double** value.

Remarks

If *object* is an instance of the **Shape** object, the **GPinY** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **GPinY** as a table parameter, use the **CDPT_GPINY** constant tag.

An instance of the parent group (parent object) can be retrieved by using the **Parent** property. To get the page which owns the shape, use the **Page** property.

The unit of measure for the coordinates are the internal ConceptDraw units (**InternalUnit**).

See Also	GPingX property, GPinY property, Page property, Parent property, Property Tags Constants

Green Property

Green Property

Gets or sets an **Integer** value, that represents the green component of an RGB color.

Applies to: Color object, ColorEntry object

Syntax

[[**Let**] *greenRet* =] *object*.**Green**

[**Let**] *object*.**Green** = *greenSet*

The **Green** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an object from the **Applies to** list.
greenRet	Optional. An **Integer** type variable.
greenSet	Required. An expression that returns an **Integer** value.

Remarks

The **Green** property is only effective if the color is an RGB color (see the **IsRGB** property).

Example

This example contains a document-level script. It demonstrates how to find out the value of the green component of the fill color (in RGB format) of a Shape object.

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
If s.FillColor.IsRGB <> false  Then     ' An RGB color?
 MsgBox(s.FillColor.Green)     ' If yes, display the value of the green
component.
endif
```

See Also Blue property, Green property, Red property, IsRGB Property

HAlign Property

A **Byte** type property. Specifies horizontal alignment of the paragraph's text with respect to the text block's alignment box.

Applies to: Paragraph object

Syntax

[**Let**] *byteRet* = *object*.**HAlign**

[**Let**] *object*.**HAlign** = *hAlignSet*

The **HAlign** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Paragraph** object.
byteRet	Optional. A **Byte** type variable.
hAlignSet	Required. An expression that returns a **Byte** value.

Remarks

There are the following types of horizontal alignment:

Constant	Value	Description
cdHorizLeft	0	Alignment to the left edge.
cdHorizCenter	1	Alignment to the center**.**
cdHorizRight	2	Alignment to the right edge.

The **HAlign** property is also a table parameter of the shape, which contains the *object* paragraph, that is, its value can be described by a formula. To work with **HAlign** as a table parameter, use the **CDPT_PARA_HALIGN** constant tag.

Example

This example demonstrates how to align the first paragraph of the shape's text to the right. It assumes a shape that contains text exists in the document.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Align the first paragraph to the right
s.Paragraph(1).HAlign = cdbHorzRight
' Inform ConceptDraw Engine about the changes for re-drawing.
s.PropertyChanged(CDPT_PARA_HALIGN)
```

See Also SetParaHAlign method

HasCharAttr Property

HasCharAttr Property

A **Boolean** type property. A flag that specifies whether the **Character** property of this style is effective. **True** - the **Character** property is effective. **False** - the character block attributes stored in the **Character** property don't apply when the style is assigned to a shape.

Applies to: Style object

Syntax

[[**Let**] *booleanRet* =] *object*.**HasCharAttr**

[**Let**] *object*.**HasCharAttr** = *hasCharAttrSet*

The **HasCharAttr** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Style** object.
booleanRet	Optional. A **Boolean** type variable.
hasCharAttrSet	Required. An expression that returns a **Boolean** value.

See Also Character property, HasEndsAttr property, HasFillAttr property, HasParaAttr property, HasPenAttr property, HasShadowAttr property, HasTxtblockAttr property

HasEndsAttr Property

HasEndsAttr Property

A **Boolean** type property. A flag that specifies whether the **LineBegin**, **LineEnd** and **LineEndSize** properties of this style, that control line end attributes, are effective. **True** - the parameters are effective. **False** - the line end parameters don't apply when the style is assigned to a shape.

Applies to: Style object

Syntax

[[**Let**] *booleanRet* =] *object*.**HasEndsAttr**

[**Let**] *object*.**HasEndsAttr** = *hasEndsAttrSet*

The **HasEndsAttr** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Style** object.
booleanRet	Optional. A **Boolean** type variable.
hasEndsAttrSet	Required. An expression that returns a **Boolean** value.

See Also	HasCharAttr property, HasFillAttr property, HasParaAttr property, HasPenAttr property, HasShadowAttr property, HasTxtblockAttr property, LineBegin property, LineEnd property, LineEndSize property

HasFillAttr Property

HasFillAttr Property

A **Boolean** type property. A flag that specifies whether the **FillColor**, **FillPatColor** and **FillPattern** properties of this style, that control the fill attributes of a shape, are effective. **True** - the parameters are effective. **False** - the fill parameters don't apply when the style is assigned to a shape.

Applies to: Style object

Syntax

[[**Let**] *booleanRet* =] *object*.**HasFillAttr**

[**Let**] *object*.**HasFillAttr** = *hasFillAttrSet*

The **HasFillAttr** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Style** object.
booleanRet	Optional. A **Boolean** type variable.
hasFillAttrSet	Required. An expression that returns a **Boolean** value.

See Also FillColor property, FillPatColor property, FillPattern property, HasCharAttr property, HasEndsAttr property, HasParaAttr property, HasPenAttr property, HasShadowAttr property, HasTxtblockAttr property

HasParaAttr Property

HasParaAttr Property

A **Boolean** type property. A flag that specifies whether the **Paragraph** property of this style is effective. **True** - the **Paragraph** property is effective. **False** - the text paragraph attributes stored in the **Paragraph** property don't apply when the style is assigned to a shape.

Applies to: Style object

Syntax

[[**Let**] *booleanRet* =] *object*.**HasParaAttr**

[**Let**] *object*.**HasParaAttr** = *hasParaAttrSet*

The **HasParaAttr** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Style** object.
booleanRet	Optional. A **Boolean** type variable.
hasParaAttrSet	Required. An expression that returns a **Boolean** value.

See Also HasCharAttr property, HasEndsAttr property, HasFillAttr property, HasPenAttr property, HasShadowAttr property, HasTxtblockAttr property, Paragraph property

HasPenAttr Property

HasPenAttr Property

A **Boolean** type property. A flag that specifies whether the **PenColor**, **PenPattern** and **PenWeight** properties of this style, that control the line attributes, are effective. **True** - the parameters are effective. **False** - the line parameters don't apply when the style is assigned to a shape.

Applies to: Style object

Syntax

[[**Let**] *booleanRet* =] *object*.**HasPenAttr**

[**Let**] *object*.**HasPenAttr** = *hasPenAttrSet*

The **HasPenAttr** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Style** object.
booleanRet	Optional. A **Boolean** type variable.
hasPenAttrSet	Required. An expression that returns a **Boolean** value.

See Also HasCharAttr property, HasEndsAttr property, HasFillAttr property, HasParaAttr property, HasShadowAttr property, HasTxtblockAttr property, PenColor property, PenPattern property, PenWeight property

HasShadowAttr Property

HasShadowAttr Property

A **Boolean** type property. A flag that specifies whether the **ShadowColor**, **ShadowPatColor** and **ShadowPattern** properties of this style, that control shadow attributes of a shape, are effective. **True** - the parameters are effective. **False** - the shadow parameters don't apply when the style is assigned to a shape.

Applies to: Style object

Syntax

[[**Let**] *booleanRet* =] *object*.**HasShadowAttr**

[**Let**] *object*.**HasShadowAttr** = *hasShadowAttrSet*

The **HasShadowAttr** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Style** object.
booleanRet	Optional. A **Boolean** type variable.
hasShadowAttrSet	Required. An expression that returns a **Boolean** value.

See Also	HasCharAttr property, HasEndsAttr property, HasFillAttr property, HasParaAttr property, HasPenAttr property, HasTxtblockAttr property, ShadowColor property, ShadowPatColor property, ShadowPattern property

HasTxtblockAttr Property

HasTxtblockAttr Property

A **Boolean** type property. A flag that specifies whether the **TextBlock** property of this style is effective. **True** - the **TextBlock** property is effective. **False** - the text block attributes stored in the **TextBlock** property don't apply when the style is assigned to a shape.

Applies to: Style object

Syntax

[[**Let**] *booleanRet* =] *object*.**HasTxtblockAttr**

[**Let**] *object*.**HasTxtblockAttr** = *hasTxtblockAttrSet*

The **HasTxtblockAttr** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Style** object.
booleanRet	Optional. A **Boolean** type variable.
hasTxtblockAttrSet	Required. An expression that returns a **Boolean** value.

See Also	HasCharAttr property, HasEndsAttr property, HasFillAttr property, HasParaAttr property, HasPenAttr property, HasShadowAttr property, TextBlock property

Height Property (Shape object)

Height Property (Shape object)

A **Double** type property. The shape's height.

Applies to: Shape object

Syntax

[[**Let**] *heightRet* =] *object*.**Height**

[**Let**] *object*.**Height** = *heightSet*

The **Height** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Window** object.
heightRet	Optional. A **Double** type variable.
heightSet	Required. An expression that returns a **Double** value.

Remarks

The **Height** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **Height** as a table parameter, use the **CDPT_HEIGHT** constant tag.

The unit of measure for the shape's height set by **Height** is the internal ConceptDraw unit (**InternalUnit**).

See Also	Angle property, GPinX property, GPinY property, FlipX property, FlipY property, Height property, LPinX property, LPinY property, Width property

Height Property (Window object)

Height Property (Window object)

Read-only. A **Long** type property. Returns the height of the window in pixels.

Applies to: Window object

Syntax

[[**Let**] *heightRet* =] *object*.**Height**

The **Height** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Window** object.
heightRet	Optional. A **Long** type variable.

Remarks

Note, that window coordinates and dimensions are measured in screen pixels. To change the position and dimensions of a window, use the **SetWindowRect** method.

See Also Left property, Top property, Width property, SetWindowRect method

HelpPath Property

HelpPath Property

Read-only. A String value. Returns the full way to files and folders which are on the way, adjusted in Preferences appendix dialogue in the Paths tab in the field of Help.

Applies to: Application object

Syntax

[[**Let**] *HelpPathRet* =] *object*.**HelpPath**

The **HelpPaht** property syntax has these Elements:

Element	**Description**

object	Required. An expression that returns a **Application** object.
HelpPathRet	Optional. A **String** type variable.

Remarks

The **HelpPath** property by default matters: **"root folder ConceptDraw DIAGRAM/Help"**.

See Also	Application object, DocumentsPath property, LibrariesPath property, TemplatesPath property

Hyperlink Property

Hyperlink Property

A **Long** type property. Represents the ID (**ID** property) of the hyperlink, associated with the shape or character block.

Applies to: Character object, Shape object

Syntax

[[**Let**] *hyperlinkIDRet* =] *object*.**Hyperlink**

[**Let**] *object*.**Hyperlink** = *hyperlinkIDSet*

The **Hyperlink** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
hyperlinkIDRet	Optional. A **Long** type variable.
hyperlinkIDSet	Required. An expression that returns a **Long** value. The ID of the hyperlink.

Remarks

If there's no hyperlink with the ID specified by *hyperlinkIDSet*, the **Hyperlink** property doesn't change its value.

The **Hyperlink** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **Hyperlink** as a table parameter, use the **CDPT_HYPERLINK** constant tag if *object* is a ConceptDraw shape, or **CDPT_CHAR_HYPERLINK** if *object* is a character block.

Example

```
dim id as Long
' Determine hyperlink ID of ShapeByID(6).Character(1)
' (assume the Shape with ID 6 exists on the page and the shape
' has a Character object, which has a hyperlink)
id = thisDoc.ActivePage.ShapeByID(6).Character(1).HyperLink
' Display the Address Property of the hyperlink with the ID obtained above.
MsgBox(thisDoc.HyperlinkByID(id).Address)
```

See Also	ID property, HyperlinkByID method, Document object, SetCharHyperlink method

ID Property

ID Property

Read-only. A **Long** type property, that indicates the ID of an object from the **Applies to** list. The ID of an object is a unique integer number, associated with the object when it's created. The ID is unique only within the scope of the collection to which it belongs. For instance, the ID of a shape is unique within the scope of the page collection of the document, however, a page with the same ID may exist in another document. The same applies to layers, etc.

Applies to: Hyperlink object, Layer object, Page object, ServObj object, Shape object, Window object

Syntax

[[**Let**] *idRet* =] *object*.**ID**

The **ID** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object in the **Applies to** list.
idRet	Optional. A **Long** type variable.

Remarks

For a **Window** object the **ID** property identifies the window. In Elementicular, in the Windows version of ConceptDraw, **ID** is a **HWND** (handle to a window) of the corresponding window.

See Also SubID property

Index Property

Index Property

Gets or sets an Integer value, which is an index representation of a color.

Applies to objects: Color

Syntax

[**Let**] *RetVal* = *object*.**Index**

[**Let**] *object*.**Index** = *SetVal*

The **Index** property syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
RetVal	An Integer value within the 0 - 255 range.
SetVal	An Integer value within the 0 - 255 range.

Remarks

Prior to using the **Index** property it's recommended that you call the IsIndex property.

Example

This example demonstrates how to find out the fill color index of the Shape object.

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
If s.FillColor.IsIndex <> false  Then    ' An index color ?
```

```
 trace s.FillColor.Index     ' If yes, display its value
endif
```

See Also Color Object, IsIndex Property

Invisible Property

Invisible Property

A Boolean value. Specifies whether the custom property is visible in the Custom Properties dialog box.

Applies to objects: CustomProp

Syntax

[[**Let**] *RetVal* =] *object*.**Invisible**

[**Let**] *object*.**Invisible** = *SetVal*

The **Invisible** property syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
RetVal	A Boolean type variable.
SetVal	A Boolean type variable.

Remarks

If **Invisible** is TRUE the corresponding **CustomProp** object won't be displayed in the Custom Properties dialog.

Example

This example demonstrates working with the **CustomProp** object.

```
Dim MyShape As Shape, MyProperty as  CustomProp
' Create a Shape
MyShape = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
'  Create custom properties  for MyShape
MyProperty = MyShape.AddCustomProp()
```

```
'  Working with the properties of MyProperty
MyProperty.Label = "IP"
MyProperty.Prompt = "TCP/IP address"
MyProperty.Type = 3
MyProperty.Format = "192.168.0.1;192.168.0.2;192.168.0.3"
MyProperty.Value = "192.168.0.1"
MyProperty.Invisible = FALSE
MyProperty.Verify = TRUE
```

See Also CustomProp Object, Document Object

Is1D Property

Is1D Property

Read-only. A **Boolean** type property. A flag that indicates whether the shape is a 1D-shape (**True**) or a 2D-shape (**False**).

Applies to: Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**Is1D**

The **Is1D** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
booleanRet	Optional. A **Boolean** type variable.

Remarks

The value of the **Is1D** property determines the behavior of the shape and which properties can be applied to the given instance of *object*. To find out the type of the shape (simple shape, group, etc.) use the **ObjType** property.

See Also ObjType property

IsBackground Property

IsBackground Property

A **Boolean** type property. A flag, specifying whether this page can be used as a background page for other pages.

Applies to: Page object

Syntax

[[**Let**] *isBackgroundRet* =] *object*.**IsBackground**

[**Let**] *object*.**IsBackground** = *isBackgroundSet*

The **IsBackground** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Page** object.
isBackgroundRet	Optional. A **Long** type variable.
isBackgroundSet	Required. An expression that returns a **Long** value.

Remarks

If **IsBackground** for a page is **False**, this page can't be used as a background page for other pages (the **BackPageID** property). A page can't have a background page associated with it if its **IsBackground** property is **True**.

See Also BackPageID property

IsCMYK Property

Read-only. Returns a **Boolean** value. If the object has a CMYK color, returns TRUE. Otherwise returns FALSE.

Applies to: Color object, ColorEntry object

Syntax

[[**Let**] *isCMYKRet* =] *object*.**IsCMYK**

The **IsCMYK** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object in the **Applies to** list.
isCMYKRet	Optional. A **Boolean** type variable.

Remarks

It's recommended that you check the status of this property before using the **Cyan**, **Magenta**, **Yellow** and **Black** properties.

Example

This example contains a document-level script. It demonstrates how to find out the color format of the fill color in a **Shape** object. If the color is in CMYK format, the value of the **cyan** component is displayed.

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
If s.FillColor.IsCMYK <> false  Then    ' A CMYK color?
 trace s.FillColor.Cyan    ' If yes, display the value of the Cyan property
endif
```

See Also	Black property, Cyan property, Magenta property, Yellow property , Index Property, IsRGB Property, IsCMYK Property

IsIndex Property

IsIndex Property

Read-only. Returns a Boolean value. If the object has an indexed color, returns TRUE. Otherwise returns FALSE.

Applies to objects: Color

Syntax

[**Let**] *RetVal* = *object*.**IsIndex**

The **IsIndex** property syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
RetVal	A Boolean type variable.

Remarks

It's recommended that you check the status of this property before using the Index Property.

Example

This example checks whether the fill color in a **Shape** object is an indexed color. If yes, it displays the index of the color.

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
If s.FillColor.IsIndex <> false  Then    ' An indexed color??
 trace s.FillColor.Index    ' If yes, display the index
endif
```

See Also Color Object, Index Property, IsRGB Property, IsCMYK Property

IsRGB Property

IsRGB Property

Read-only. Returns a **Boolean** value. If the object has an RGB color, returns TRUE. Otherwise returns FALSE.

Applies to: Color object, ColorEntry object

Syntax

[[**Let**] *isRGBRet* =] *object*.**IsRGB**

The **IsRGB** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object in the **Applies to** list.
isRGBRet	Optional. A **Boolean** type variable.

Remarks

It's recommended that you check the status of this property before using the Red, Green, and Blue properties.

Example

This example contains a document-level script. It demonstrates how to find out the color format of the fill color in a **Shape** object. If the color is in the RGB format, the value of the **Red** component is displayed.

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
If s.FillColor.IsRGB <> false  Then    ' An  RGB color ?
 trace s.FillColor.Red    ' If yes, display the value of the Red property
endif
```

See Also	Blue property, Green property, Red property, Index Property, IsRGB Property, IsCMYK Property

IsTransparent Property

IsTransparent Property

A **Boolean** value. If **True**, the color is transparent. Otherwise is **False**.

Applies to: Color object, ColorEntry object

Syntax
[[**Let**] *transparentRet* =] *object*.**IsTransparent**

[**Let**] *object*.**IsTransparent** = *transparentSet*

The **IsTransparent** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object in the **Applies to** list.
transparentRet	Optional. A **Boolean** type variable.
transparentSet	Required. An expression that returns a **Boolean** value.

Example

This example contains a document-level script. It demonstrates how to make a shape's fill transparent.

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
If s.FillColor.IsTransparent <> true  Then    ' Is the color transparent?
 s.FillColor.IsTransparent = true    ' If not, make it transparent
 s.PropertyChanged(CDPT_FILLCOLOR)   ' Inform ConceptDraw Engine
endif
```

See Also	Index Property, IsRGB Property, IsCMYK Property

Label Property

Label Property

A **String** type property. The label of a custom property.

Applies to: CustomProp object

Syntax
[[**Let**] *labelStrRet* =] *object*.**Label**

[**Let**] *object*.**Label** = *labelStrSet*

The **Label** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **CustomProp** object.
labelStrRet	Optional. A **String** type variable.
labelStrSet	Required. An expression that returns a **String** value.

Example

This example demonstrates working with the **CustomProp** object.

```
Dim MyShape As Shape, MyProperty as  CustomProp
' Create a Shape
MyShape = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
'  Create custom properties  for MyShape
MyProperty = MyShape.AddCustomProp()
'  Working with the properties of MyProperty
MyProperty.Label = "IP"
MyProperty.Prompt = "TCP/IP address"
MyProperty.Type = 3
MyProperty.Format = "192.168.0.1;192.168.0.2;192.168.0.3"
MyProperty.Value = "192.168.0.1"
MyProperty.Invisible = FALSE
MyProperty.Verify = TRUE
```

Language Property

Language Property

A **Byte** type property. Gets or sets the charset of the character block.

Applies to: Character object

Syntax

[[**Let**] *byteRet* =] *object*.**Language**

[**Let**] *object*.**Language** = *languageSet*

The **Language** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Character** object.
byteRet	Optional. A **Byte** type variable.
languageSet	Required. An expression that returns a **Byte** value.

Remarks

The **Language** property can take of the the following possible values:

Constant	**Value**	**Description**
ANSI_CHARSET	0	ANSI charset.
DEFAULT_CHARSET	1	Default charset.
SYMBOL_CHARSET	2	Symbol charset.
MAC_CHARSET	77	Macintosh charset.
SHIFTJIS_CHARSET	128	charset.
HANGEUL_CHARSET	129	Hungarian charset.
HANGUL_CHARSET	129	Hungarian charset.
JOHAB_CHARSET	130	charset.
GB2312_CHARSET	134	charset.
CHINESEBIG5_CHARSET	136	Chinese charset.
GREEK_CHARSET	161	Greek charset.
TURKISH_CHARSET	162	Turkish charset.
VIETNAMESE_CHARSET	163	Vietnamese charset.
HEBREW_CHARSET	177	Hebrew charset.
ARABIC_CHARSET	178	Arabic charset.
BALTIC_CHARSET	186	Baltic charset.
RUSSIAN_CHARSET	204	Russian (cyrillic) charset.
THAI_CHARSET	222	Thai charset.
EASTEUROPE_CHARSET	238	East Europe charset.
OEM_CHARSET	255	OEM charset.

The **Language** property is also a table parameter of the shape which contains the *object* character block, that is, its value can be described by a formula. To work with **Language** as a table parameter, use the **CDPT_CHAR_LANGUAGE** constant tag.

Example

This example demonstrates how to find out the charset of the first Character object of the Shape object with ID 1, located on the current page.

```
Dim MyShape As Shape
' The shape with ID 1 must exist on the current page
Set MyShape = thisDoc.ActivePage.ShapeByID(1)
' Display information about the charset of MyShape.Character(1)
' (it assumes MyShape.Character(1) exists)
MsgBox(MyShape.Character(1).Language)
```

See Also SetCharLanguage method

Layer Property

Layer Property

A **Long** type property. ID of the layer (the **ID** property), on which the shape/service object is located.

Applies to: ServObj object, Shape object

Syntax

[[**Let**] *layerIDRet* =] *object*.**Layer**

[**Let**] *object*.**Layer** = *layerIDSet*

The **Layer** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
layerIDRet	Optional. A **Long** type variable.
layerIDSet	Required. An expression that returns a **Long** value. The ID of the layer.

Remarks

If there is no layer with the specified *layerIDSet* in the layer collection of the document, the **Layer** property doesn't change its value, as a shape can't be located on a nonexisting layer. To check whether the layer with the specified ID exists in the document, use the **LayerByID** method.

See Also ID property, LayerByID method, Layer object

LeftInd Property

LeftInd Property

A **Singe** type property. The distance all lines of text in a paragraph are indented from the left margin of the text block.

Applies to: Paragraph object

Syntax

[**Let**] *singleRet* = *object*.**LeftInd**

[**Let**] *object*.**LeftInd** = *leftIndSet*

The **LeftInd** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Paragraph** object.
singleRet	Optional. A **Single** type variable.
leftIndSet	Required. An expression that returns a **Single** value.

Remarks

Indents are specified in internal ConceptDraw units (**InternalUnit**).

The **LeftInd** property is also a table parameter of the shape that contains the *object* paragraph, that is, its value can be described by a formula. To work with **LeftInd** as a table parameter, use the **CDPT_PARA_LEFTIND** constant tag.

Example

This example demonstrates how to set a left indent for the second paragraph of a shape. It assumes there is a shape on the current page, and its text contains at least two paragraphs.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
```

```
' Move the second paragraph of by 100 points right from the left border of the
text block.
s.Paragraph(2).LeftInd = 100
' Inform ConceptDraw Engine about the changes for re-drawing.
s.PropertyChanged(CDPT_PARA_LEFTIND)
```

See Also SetParaLeftInd method

LeftMargin Property

LeftMargin Property

A **Single** type property. The distance the text inside the text block is offset from the left border of the text box.

Applies to: TextBlock object

Syntax

[[**Let**] *singleRet* =] *object*.**LeftMargin**

[**Let**] *object*.**LeftMargin** = *leftMarginSet*

The **LeftMargin** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **TextBlock** object.
singleRet	Optional. A **Single** type variable.
leftMarginSet	Required. An expression that returns a **Single** value.

Remarks

The unit of measure for the **LeftMargin** property are internal ConceptDraw units (**InternalUnit**).

The **LeftMargin** property is also a table parameter of the shape which contains the *object* text block, that is, its value can be described by a formula. To work with **LeftMargin** as a table parameter, use the **CDPT_LEFTMARGIN** constant tag.

Example

This example demonstrates how to increase the distance between the text and left border of the text block of an existing shape.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Increase the distance between the text and the left border of the text box
by 20 points.
s.TextBlock.LeftMargin = 20
' Inform ConceptDraw Engine about the changes for re-drawing.
s.PropertyChanged(CDPT_LEFTMARGIN)
```

Left Property

Left Property

For the DRect object:
Gets or sets a Double value, that represents the coordinates of the leftmost point of the instance of the object.

For the Window object:
Gets or sets a Long value, that represents the coordinates of the leftmost point of the window, associated with the object.

Applies to objects: DRect, Window

Syntax

For the **DRect** object:
[[**Let**] *RetVal* =] *object*.**Left**

[**Let**] *object*.**Left** = *SetVal*

Element	**Description**
object	A reference to an instance of the object.
RetVal	A Double type variable.
SetVal	A Double value.

For the **Window** object:
[[**Let**] *RetVal* =] *object*.**Left**

Element	**Description**
object	A reference to an instance of the object.
RetVal	A Long type variable. The default unit of measure is unit.

Example

```
Dim MyObject as new DRect   ' Create an instance of the object
MyObject.Left = 100
```

See Also

DRect Object, Window Object

Left Property (Window object)

Left Property (Window object)

Read-only. A **Long** type property. Returns the X-coordinate of the top left corner of the window.

Applies to: Window object

Syntax

[[**Let**] *leftRet* =] *object*.**Left**

The **Left** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Window** object.
leftRet	Optional. A **Long** type variable.

Remarks

Note, that the coordinates of the window position are specified in screen pixels, and the coordinate origin is in the left top corner of the parent window frame. To change the dimensions and position of the window, use the **SetWindowRect** method.

Example

```
. . . . . . . . . .
```

See Also	Top property, Height property, Width property, SetWindowRect method

LibrariesPath Property

LibrariesPath Property

Read-only. A **String** value. Returns the full way to files which are on the way, adjusted in **Preferences** appendix dialogue in the **Paths** tab in the field of **Libraries**.

Applies to: Application object

Syntax

[[**Let**] *LibrariesPathRet* =] *object*.**LibrariesPath**

The **LibrariesPath** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Application** object.
LibrariesPathRet	Optional. A **String** type variable.

Remarks

The **LibrariesPath** property by default matters: **"personal folder of the user /AppData/Local/CS Odessa/ConceptDraw Office/ConceptDraw DIAGRAM DIAGRAM/Libraries"**.

See Also	Application object,DocumentsPath property, HelpPath property, TemplatesPath property

Library Property

Library Property

Read-only. Gets an instance of the **Library** object that represents the active library in this library window.

Applies to: Window object

Syntax

[[**Set**] *libraryRet* =] *object*.**Library**

The **Library** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Window** object.
libraryRet	Optional. A **Library** type variable.

Remarks

This property is only effective if the window is a library window (see the **Type** property). For all other window types the **Library** property returns **Nothing**. The active library is the library whose contents is displayed in the window.

See Also Document property, Page property, Shape property, Library object

LineBegin Property

LineBegin Property

A **Long** type property. Specifies the begin arrowhead type for a 1D-shape.

Applies to: Shape object, Style object

Syntax

[[**Let**] *longRet* =] *object*.**LineBegin**

[**Let**] *object*.**LineBegin** = *lineBeginSet*

The **LineBegin** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
longRet	Optional. A **Long** type variable.
lineBeginSet	Required. An expression that returns a **Long** value.

Remarks

The value of **LineBegin** can be in the range of **0** to **60**. The **0** value means the 1D-shape has no begin arrowhead (No Arrows).

Shape object:
The **LineBegin** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LineBegin** as a table parameter, use the **CDPT_LINEBEGIN** constant tag.

Style object:
When a style is assigned to a shape, the parameters of the **LineBegin** property of the style are set to the **LineBegin** property of the shape. **LineBegin** is only effective when the **HasEndsAttr** property of this style is **True**.

See Also HasEndsAttr property, LineEnd property, LineEndSize property

LineEndSize Property

LineEndSize Property

A **Long** type property. Specifies the begin and end arrowhead size for a 1D-shape.

Applies to: Shape object, Style object

Syntax

[[**Let**] *longRet* =] *object*.**LineEndSize**

[**Let**] *object*.**LineEndSize** = *lineEndSizeSet*

The **LineEndSize** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
longRet	Optional. A **Long** type variable.
lineEndSizeSet	Required. An expression that returns a **Long** value.

Remarks

The **LineEndSize** property can take the following values:

Constant	Value	Description
cdLESTiny	0	Tiny size.
cdLESSmall	1	Small size.
cdLESMedium	2	Medium size.
cdLESLarge	3	Large size.
cdLESHuge	4	Huge size.

Shape object:
The **LineEndSize** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LineEndSize** as a table parameter, use the **CDPT_LINEENDSIZE** constant tag.

Style object:
When a style is assigned to a shape, the parameters of the **LineEndSize** property of the style are set to the **LineEndSize** property of the shape. **LineEndSize** is only effective when the **HasEndsAttr** property of this style is **True**.

See Also HasEndsAttr property, LineBegin property, LineEnd property

LineEnd Property

LineEnd Property

A **Long** type property. Specifies the end arrowhead type for a 1D-shape.

Applies to: Shape object, Style object

Syntax

[[**Let**] *longRet* =] *object*.**LineEnd**

[**Let**] *object*.**LineEnd** = *lineEndSet*

The **LineEnd** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
longRet	Optional. A **Long** type variable.
lineEndSet	Required. An expression that returns a **Long** value.

Remarks

The value of **LineEnd** can be in the range of **0** to **60**. The **0** value means the 1D-shape has no end arrowhead (No Arrows).

Shape object:
The **LineEnd** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LineEnd** as a table parameter, use the **CDPT_LINEEND** constant tag.

Style object:
When a style is assigned to a shape, the parameters of the **LineEnd** property of the style are set to the **LineEnd** property of the shape. **LineEnd** is only effective when the **HasEndsAttr** property of this style is **True**.

See Also HasEndsAttr property, LineBegin property, LineEndSize property

ConnectCrossType Property

LineJumpOrient Property

A **Long** type property. Gets and sets the orientation of the line jumps on smart connector's crossings.

Applies to: Document object

Syntax

[[**Let**] *jumpOrientRet* =] *object*.**LineJumpOrient**

[**Let**] *object*.**LineJumpOrient** = *jumpOrientSet*

The **LineJumpOrient** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
jumpOrientRet	Optional. A **Long** type variable.
jumpOrientSet	Required. An expression that returns a **Long** value.

Remarks

The **LineJumpOrient** can take only one of these values:

Constant	**Value**	**Description**
cdNoJumps	0	No line jumps
cdVertJumps	1	Vertical line jumps orientation
cdHorisJumps	2	Vertical line jumps orientation

You can also change the **LineJumpOrient** property from the ConceptDraw menu:"Tools->Line Jump Orientation".

See Also	FlowAroundObjects property, LineJumpSize property, LineJumpType property, MaxNumberOfLegs property, MinDistToShapes property, PassThroughGroups property

LineJumpSize Property

LineJumpSize Property

A **Double** type property. Gets and sets the line jump size for smart connector's crossings.

Applies to: Document object

Syntax

[[**Let**] *jumpSizeRet* =] *object*.**LineJumpSize**

[**Let**] *object*.**LineJumpSize** = *jumpSizeSet*

The **LineJumpSize** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Document** object.
jumpSizeRet	Optional. A **Double** type variable.
jumpSizeSet	Required. An expression that returns a **Double** value.

Remarks

The value of the line jump size is set in the internal units of ConceptDraw (**InternalUnit**).

The value of the **LineJumpSize** property can be also viewed and changed in the following dialog in ConceptDraw:"File->Document Properties->Advanced->Connectors And Routing".

See Also	FlowAroundObjects property, LineJumpOrient property, LineJumpType property, MaxNumberOfLegs property, MinDistToShapes property, PassThroughGroups property

LineJumpType Property

LineJumpType Property

A **Long** type property. Gets and sets the type of smart connector's crossings in the document.

Applies to: Document object

Syntax

[[**Let**] *jumpTypeRet* =] *object*.**LineJumpType**

[**Let**] *object*.**LineJumpType** = *jumpTypeSet*

The **LineJumpType** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Document** object.
jumpTypeRet	Optional. A **Long** type variable.
jumpTypeSet	Required. An expression that returns a **Long** value.

Remarks

The **LineJumpType** can take only one of these values:

Constant	**Value**	**Description**
cdJumpSquare	1	As a square bridge
cdJumpArc	2	As an arcover bridge
cdJump2sides	3	As a two-side bridge
cdJump3sides	4	As a three-side bridge
cdJumpGap	5	As a gap

You can also change the **LineJumpOrient** property from the ConceptDraw menu: "Tools->Line Jump Type".

See Also FlowAroundObjects property, LineJumpOrient property, LineJumpSize property, MaxNumberOfLegs property, MinDistToShapes property, PassThroughGroups property

LineSpacing Property

LineSpacing Property

A **Single** type property. Specifies the distance between the lines of the paragraph.

Applies to: Paragraph object

Syntax

[**Let**] *singleRet* = *object*.**LineSpacing**

[**Let**] *object*.**LineSpacing** = *lineSpacingSet*

The **LineSpacing** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Paragraph** object.
singleRet	Optional. A **Single** type variable.
lineSpacingSet	Required. An expression that returns a **Single** value.

Remarks

The line spacing is specified in internal ConceptDraw units (**InternalUnit**).

The **LineSpacing** property is also a table parameter of the shape which contains the *object* paragraph, that is, its value can be described by a formula. To work with **LineSpacing** as a table parameter, use the **CDPT_PARA_LINESPACING** constant tag.

Example

This example demonstrates how to increase the spacing between the lines in a paragraph of text. It assumes there is a shape which contains text on the active page of the document.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Set line spacing.
s.Paragraph(1).LineSpacing = 100
' Inform ConceptDraw Engine about the changes for re-drawing
s.PropertyChanged(CDPT_PARA_LINESPACING)
```

See Also	SetParaLineSpacing method

LinkType Property

LinkType Property

Read-only. A **Long** value, indicating the type of the hyperlink.

Applies to: Hyperlink object

Syntax

[[**Let**] *linkTypeRet* =] *object*.**LinkType**

The **LinkType** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Hyperlink** object.
linkTypeRet	Optional. A **Long** type variable.

Remarks

The **LinkType** can have the following values:

Constant	**Value**	**Description**

cdLinkNone	0	The **Hyperlink** object has no hyperlink (the hyperlink doesn't point to anything).
cdLinkToFile	1	The hyperlink points to a local file, which is either a ConceptDraw document, or a file of any other supported format. If the hyperlink points to a ConceptDraw document, it may also indicate a page and a shape inside this document, which are described by the **PageID** and **ShapeID** properties respectively.
cdLinkToURL	2	The hyperlink points to an Internet address (URL).
cdLinkToPageShape	3	The hyperlink points to a page or a shape inside the same document. The page and the shape are described by the **PageID** and **ShapeID** properties respectively.

See Also ID property, PageID property, ShapeID property

LocalPath Property

LocalPath Property

Read-only. A **Boolean** value. A flag that specifies in which form the path to the file (the **Address** property), pointed to by the hyperlink, is stored.

Applies to: Hyperlink object

Syntax

[[**Let**] *localPathRet* =] *object*.**LocalPath**

The **LocalPath** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Hyperlink** object.
localPathRet	Optional. A **Boolean** type variable.

Remarks

If **LocalPath** is **True**, the **Address** property contains the relative path to the file (with respect to the folder which was the current folder when the hyperlink was created). Otherwise, the full path to the file is stored. (absolute path).

The **LocalPath** property is only effective when the hyperlink points to a file, that is, the hyperlink is of the **cdLinkToFile** type (see the LinkType property).

See Also Address property, LinkType property

LockAspect Property

LockAspect Property

A **Boolean** type property. A flag that specifies whether the shape can be resized unporportionally.**True** - only proportional resizing is the width and height of the shape is allowed. **False** - both proportional and unproportional resizing is possible.

Applies to: Shape object

Syntax

[[**Let**] *lockAspectRet* =] *object*.**LockAspect**

[**Let**] *object*.**LockAspect** = *lockAspectSet*

The **LockAspect** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
lockAspectRet	Optional. A **Boolean** type variable.
lockAspectSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockAspect** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockAspect** as a table parameter, use the **CDPT_LOCKASPECT** constant tag.

See Also LockBegin property, LockCalcWH property, LockDelete property, LockEnd property, LockFlipX property, LockFlipY property, LockHeight property, LockMoveX property, LockMoveY property, LockRotate property, LockTextBound property, LockVertex property, LockWidth property

LockBegin Property

LockBegin Property

A **Boolean** type property. A flag that specifies whether the begin point of the 1D-shape can be repositioned with the mouse.**True** - it can't be repositioned. **False** - it can be repositioned.

Applies to: Shape object

Syntax

[[**Let**] *lockBeginRet* =] *object*.**LockBegin**

[**Let**] *object*.**LockBegin** = *lockBeginSet*

The **LockBegin** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
lockBeginRet	Optional. A **Boolean** type variable.
lockBeginSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockBegin** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockBegin** as a table parameter, use the **CDPT_LOCKBEGIN** constant tag.

See Also BeginX property, BeginY property, LockAspect property, LockCalcWH property, LockDelete property, LockEnd property, LockFlipX property, LockFlipY property, LockHeight property, LockMoveX property, LockMoveY property, LockRotate property, LockTextBound property, LockVertex property, LockWidth property

LockCalcWH Property

LockCalcWH Property

A **Boolean** type property. A flag that specifies whether to re-calculate the size of the shape's alignment box, when the coordinates of the shape's vertices were modified. **True** - the alignment box size is re-calculated, **False** - not re-calculated.

Applies to: Shape object

Syntax

[[**Let**] *lockCalcWHRet* =] *object*.**LockCalcWH**

[**Let**] *object*.**LockCalcWH** = *lockCalcWHSet*

The **LockCalcWH** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
lockCalcWHRet	Optional. A **Boolean** type variable.
lockCalcWHSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockCalcWH** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockCalcWH** as a table parameter, use the **CDPT_LOCKCALCWH** constant tag.

See Also	LockAspect property, LockBegin property, LockDelete property, LockEnd property, LockFlipX property, LockFlipY property, LockHeight property, LockMoveX property, LockMoveY property, LockRotate property, LockTextBound property, LockVertex property, LockWidth property

LockConnector Property

LockConnector Property

A **Boolean** type property. A flag that blocks a smart connector from re-routing when its end points are repositioned. **True** - the smart connector doesn't change its form if it's possible, **False** - the smart connector changes its form automatically.

Applies to: Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**LockConnector**

[**Let**] *object*.**LockConnector** = *lockConnectorSet*

The **LockConnector** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
booleanRet	Optional. A **Boolean** type variable.
lockConnectorSet	Required. An expression that returns a **Boolean** value.

Remarks

For a new smart connector the **LockConnector** property is **False** by default.

See Also	LockAspect property, LockBegin property, LockDelete property, LockEnd property, LockFlipX property, LockFlipY property, LockHeight property, LockMoveX property, LockMoveY property, LockRotate property, LockTextBound property, LockVertex property, LockWidth property

LockDelete Property

LockDelete Property

A **Boolean** type property. A flag that specifies whether the shape is protected from deleting.**True** - protection is on, **False** - protection is off.

Applies to: Shape object

Syntax

[[**Let**] *lockDeleteRet* =] *object*.**LockDelete**

[**Let**] *object*.**LockDelete** = *lockDeleteSet*

The **LockDelete** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
lockDeleteRet	Optional. A **Boolean** type variable.
lockDeleteSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockDelete** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockDelete** as a table parameter, use the **CDPT_LOCKDELETE** constant tag.

See Also	LockAspect property, LockBegin property, LockCalcWH property, LockEnd property, LockFlipX property, LockFlipY property, LockHeight property, LockMoveX property, LockMoveY property, LockRotate property, LockTextBound property, LockVertex property, LockWidth property

Locked Property

Locked Property

A Boolean value. Specifies whether the layer is locked.

Applies to objects: Layer

Syntax

[[**Let**] *RetVal* =] *object*.**Locked**

[**Let**] *object*.**Locked** = *SetVal*

The **Locked** property syntax has these Elements:

Element	Description
object	A reference to an instance of the object.
RetVal	A Boolean type variable.
SetVal	A Boolean type value.

Remarks

If **Locked** is TRUE, you won't be able to edit shapes on this layer. You can set the **Locked** property to TRUE only when the layer is not active.

Example

```
Dim MyLayer as Layer
' Get the second Layer of thisDoc
set MyLayer = thisDoc.Layer(2)
' Make it Locked
' (assume that MyLayer is not active)
MyLayer.Locked = True
```

See Also Layer Object, Document Object

LockEnd Property

LockEnd Property

A **Boolean** type property. A flag that specifies whether the end point of the 1D-shape can be repositioned with the mouse.**True** - it can't be repositioned. **False** - it can be repositioned.

Applies to: Shape object

Syntax

[[**Let**] *lockEndRet* =] *object*.**LockEnd**

[**Let**] *object*.**LockEnd** = *lockEndSet*

The **LockEnd** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.

lockEndRet	Optional. A **Boolean** type variable.
lockEndSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockEnd** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockEnd** as a table parameter, use the **CDPT_LOCKEND** constant tag.

See Also	EndX property, EndY property, LockAspect property, LockBegin property, LockCalcWH property, LockDelete property, LockFlipX property, LockFlipY property, LockHeight property, LockMoveX property, LockMoveY property, LockRotate property, LockTextBound property, LockVertex property, LockWidth property

LockFlipX Property

LockFlipX Property

A **Boolean** type property. A flag that specifies whether the shape can be flipped horizontally. **True** - horizontal flipping is allowed. **False** - horizontal flipping is not allowed.

Applies to: Shape object

Syntax

[[**Let**] *lockFlipXRet* =] *object*.**LockFlipX**

[**Let**] *object*.**LockFlipX** = *lockFlipXSet*

The **LockFlipX** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
lockFlipXRet	Optional. A **Boolean** type variable.
lockFlipXSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockFlipX** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockFlipX** as a table parameter, use the **CDPT_LOCKFLIPX** constant tag.

See Also	FlipX property, LockAspect property, LockBegin property, LockCalcWH property, LockDelete property, LockEnd property, LockFlipY property, LockHeight property, LockMoveX property, LockMoveY property, LockRotate property, LockTextBound property, LockVertex property, LockWidth property

LockFlipY Property

LockFlipY Property

A **Boolean** type property. A flag that specifies whether the shape can be flipped vertically. **True** - vertical flipping is allowed. **False** - vertical flipping is not allowed.

Applies to: Shape object

Syntax

[[**Let**] *lockFlipYRet* =] *object*.**LockFlipY**

[**Let**] *object*.**LockFlipY** = *lockFlipYSet*

The **LockFlipY** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
lockFlipXRet	Optional. A **Boolean** type variable.
lockFlipXSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockFlipY** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockFlipY** as a table parameter, use the **CDPT_LOCKFLIPY** constant tag.

See Also	FlipY property, LockAspect property, LockBegin property, LockCalcWH property, LockDelete property, LockEnd property, LockFlipX property, LockHeight property, LockMoveX property, LockMoveY property, LockRotate property, LockTextBound property, LockVertex property, LockWidth property

LockHeight Property

LockHeight Property

A **Boolean** type property. A flag that specifies whether to allow changing the shape's height when the shape is resized. **True** - height is protected from resizing, **False** - height is not protected from resizing.

Applies to: Shape object

Syntax

[[**Let**] *lockHeightRet* =] *object*.**LockHeight**

[**Let**] *object*.**LockHeight** = *lockHeightSet*

The **LockHeight** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
lockHeightRet	Optional. A **Boolean** type variable.
lockHeightSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockHeight** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockHeight** as a table parameter, use the **CDPT_LOCKHEIGHT** constant tag.

See Also	Height property (Shape object), LockAspect property, LockBegin property, LockCalcWH property, LockDelete property, LockEnd property, LockFlipX property, LockFlipY property, LockMoveX property, LockMoveY property, LockRotate property, LockTextBound property, LockVertex property, LockWidth property

LockMoveX Property

LockMoveX Property

A **Boolean** type property. A flag that protects the shape from repositioning horizontally. **True** - the shape can't be repositioned horizontally, **False** - horizontal repositioning is allowed.

Applies to: Shape object

Syntax

[[**Let**] *lockMoveXRet* =] *object*.**LockMoveX**

[**Let**] *object*.**LockMoveX** = *lockMoveXSet*

The **LockMoveX** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
lockMoveXRet	Optional. A **Boolean** type variable.
lockMoveXSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockMoveX** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockMoveX** as a table parameter, use the **CDPT_LOCKMOVEX** constant tag.

See Also	GPinX property, LockAspect property, LockBegin property, LockCalcWH property, LockDelete property, LockEnd property, LockFlipX property, LockFlipY property, LockHeight property, LockMoveY property, LockRotate property, LockTextBound property, LockVertex property, LockWidth property

LockMoveY Property

LockMoveY Property

A **Boolean** type property. A flag that protects the shape from repositioning vertically. **True** - the shape can't be repositioned vertically, **False** - vertical repositioning is allowed.

Applies to: Shape object

Syntax

[[**Let**] *lockMoveYRet* =] *object*.**LockMoveY**

[**Let**] *object*.**LockMoveY** = *lockMoveYSet*

The **LockMoveY** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
lockMoveYRet	Optional. A **Boolean** type variable.
lockMoveYSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockMoveY** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockMoveY** as a table parameter, use the **CDPT_LOCKMOVEY** constant tag.

See Also	GPinY property, LockAspect property, LockBegin property, LockCalcWH property, LockDelete property, LockEnd property, LockFlipX property, LockFlipY property, LockHeight property, LockMoveX property, LockRotate property, LockTextBound property, LockVertex property, LockWidth property

LockRotate Property

LockRotate Property

A **Boolean** type property. A flag that protects the shape from rotating. **True** - the shape can't be rotated, **False** - rotation is allowed.

Applies to: Shape object

Syntax
[[**Let**] *lockRotateRet* =] *object*.**LockRotate**

[**Let**] *object*.**LockRotate** = *lockRotateSet*

The **LockRotate** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
lockRotateRet	Optional. A **Boolean** type variable.
lockRotateSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockRotate** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockRotate** as a table parameter, use the **CDPT_LOCKROTATE** constant tag.

See Also	Angle property, LockAspect property, LockBegin property, LockCalcWH property, LockDelete property, LockEnd property, LockFlipX property, LockFlipY property, LockHeight property, LockMoveX property, LockMoveY property, LockTextBound property, LockVertex property, LockWidth property

LockTextBound Property

LockTextBound Property

A **Boolean** type property. Whether A flag that specifies can overstep the bounds of limit of the text of object of object. **True** - limits of the text can't overstep the bounds of object. **False** - limits of the text can overstep the bounds of object.

Applies to: Shape object

Syntax
[[**Let**] *lockTextBoundRet* =] *object*.**LockTextBound**

[**Let**] *object*.**LockTextBound** = *lockTextBoundSet*

The **LockTextBound** property syntax has these Elements:

Element	**Description**

object	Required. An expression that returns a **Shape** object.
lockTextBoundRet	Optional. A **Boolean** type variable.
lockTextBoundSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockTextBound** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockTextBound** as a table parameter, use the **CDPT_LOCKTEXTBOUND** constant tag.

See Also LockAspect property, LockBegin property, LockCalcWH property, LockDelete property, LockEnd property, LockFlipX property, LockFlipY property, LockHeight property, LockMoveX property, LockMoveY property, LockRotate property, LockVertex property, LockWidth property

LockVertex Property

LockVertex Property

A **Boolean** type property. A flag that protects vertices from editing with the mouse.**True** - protection is on, **False** - protection is off.

Applies to: Shape object

Syntax

[[**Let**] *lockVertexRet* =] *object*.**LockVertex**

[**Let**] *object*.**LockVertex** = *lockVertexSet*

The **LockVertex** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
lockBeginRet	Optional. A **Boolean** type variable.
lockBeginSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockVertex** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockVertex** as a table parameter, use the **CDPT_LOCKVERTEX** constant tag.

See Also	LockAspect property, LockBegin property, LockCalcWH property, LockDelete property, LockEnd property, LockFlipX property, LockFlipY property, LockHeight property, LockMoveX property, LockMoveY property, LockRotate property, LockTextBound property, LockWidth property

LockWidth Property

LockWidth Property

A **Boolean** type property. A flag that specifies whether to allow changing the shape's width when the shape is resized. **True** - width is protected from resizing, **False** - width is not protected from resizing.

Applies to: Shape object

Syntax

[[**Let**] *lockWidthRet* =] *object*.**LockWidth**

[**Let**] *object*.**LockWidth** = *lockWidthSet*

The **LockWidth** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
lockBeginRet	Optional. A **Boolean** type variable.
lockBeginSet	Required. An expression that returns a **Boolean** value.

Remarks

The **LockWidth** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LockWidth** as a table parameter, use the **CDPT_LOCKWIDTH** constant tag.

See Also LockAspect property, LockBegin property, LockCalcWH property, LockDelete property, LockEnd property, LockFlipX property, LockFlipY property, LockHeight property, LockMoveX property, LockMoveY property, LockRotate property, LockTextBound property, LockVertex property, Width property (Shape object)

LPinX Property

LPinX Property

A **Double** type property. The X offset of the shape's rotation center from the center of the shape's coordinate system.

Applies to: Shape object

Syntax

[[**Let**] *lpinXRet* =] *object*.**LPinX**

[**Let**] *object*.**LPinX** = *lpinXSet*

The **LPinX** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
lpinXRet	Optional. A **Double** type variable.
lpinXSet	Required. An expression that returns a **Double** value.

Remarks

The **LPinX** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LPinX** as a table parameter, use the **CDPT_LPINX** constant tag.

Note, that modifying the **LPinX** automatically changes the value of the **GPinX** property. The unit of measure for the offset are internal ConceptDraw units (**InternalUnit**).

See Also GPinX property, GPinY property, LPinY property

LPinY Property

LPinY Property

A **Double** type property. The Y offset of the shape's rotation center from the center of the shape's coordinate system.

Applies to: Shape object

Syntax

[[**Let**] *lpinYRet* =] *object*.**LPinY**

[**Let**] *object*.**LPinY** = *lpinYSet*

The **LPinY** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
lpinYRet	Optional. A **Double** type variable.
lpinYSet	Required. An expression that returns a **Double** value.

Remarks

The **LPinY** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **LPinY** as a table parameter, use the **CDPT_LPINY** constant tag.

Note, that modifying the **LPinY** automatically changes the value of the **GPinY** property. The unit of measure for the offset are internal ConceptDraw units (**InternalUnit**).

See Also GPinX property, GPinY property, LPinX property

Magenta Property

Gets or sets an **Integer** value, that represents the magenta component of CMYK color.

Applies to: Color object, ColorEntry object

Syntax

[[**Let**] *magentaRet* =] *object*.**Magenta**

[**Let**] *object*.**Magenta** = *magentaSet*

The **Magenta** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object from the **Applies to** list.
magentaRet	Optional. An **Integer** value.
magentaSet	Required. An expression that returns an **Integer** value.

Remarks

The **Magenta** property is only effective if the color is a CMYK color (see the **IsCMYK** property).

Example

This example contains a document-level script. It demonstrates how to find out the value of the magenta component of the fill color (in CMYK format) of a Shape object.

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
If s.FillColor.IsCMYK<> false  Then    ' A CMYK color?
 MsgBox(s.FillColor.Magenta)    ' If yes, display the value of the magenta
component.
endif
```

See Also Cyan property, Yellow property, Black property, IsCMYK property

MaxNumOfLegs Property

MaxNumOfLegs Property

A **Long** type property. Gets or sets the maximum possible number of legs for all smart connectors of the document.

Applies to: Document object

Syntax

[[**Let**] *numOfLegsRet* =] *object*.**MaxNumOfLegs**

[**Let**] *object*.**MaxNumOfLegs** = *numOfLegsSet*

The **MaxNumOfLegs** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
numOfLegsRet	Optional. A **Long** type variable.
numOfLegsSet	Required. An expression that returns a **Long** value. The new maximum number of legs for smart connectors of the document.

Remarks

If **MaxNumOfLegs** equals to **0**, the number of legs for all smart connectors of the document is unlimited. The minimal number of smart connector legs is **3**, so any value of *numOfLegsSet* less than **3** is equivalent to setting the unlimited number of legs. This property can also be changed from within ConceptDraw, using the menu "File->Document Properties->Advanced->Connectors And Routing".

See Also	FlowAroundObjects property, LineJumpOrient property, LineJumpSize property, LineJumpType property, MinDistToShapes property, PassThroughGroups property

Menu Property

Menu Property

A String value. Gets or sets a menu item name.

Applies to objects: Action

Syntax

[**Let**] *RetVal* = *object*.**Menu**

[**Let**] *object*.**Menu** = *SetVal*

The **Menu** property syntax has these Elements:

Element	Description
object	A reference to an instance of the object.
RetVal	A String value that specifies the name of a menu item.
SetVal	A String value that specifies the name of a menu item.

Example

This example demonstrates using the **Menu** property.

```
Dim s as Shape, MyAction as Action
' Assume Shape with ID 1 exists on the active page.
' Assume the Shape contains at least one Action
s = thisDoc.ActivePage.ShapeByID(1)
' Get reference to an instance of the Action object
Set MyAction = s.Action(1)
' Sets the name of the action in the menu
MyAction.Menu = "Hide Shape"
```

See Also	Action Object, Shape Object, SetPropertyFormula Method, ActionsNum Method, AddAction Method, Action Method, RemoveAction Method

MinDistToShapes Property

MinDistToShapes Property

A **Double** type property. Sets the minimal distance between a smart connector and other shapes on the same page. Applies to all smart connectors of the document.

Applies to: Document object

Syntax
[[**Let**] *minDistanceRet* =] *object*.**MinDistToShapes**

[**Let**] *object*.**MinDistToShapes** = *minDistanceSet*

The **MinDistToShapes** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
minDistanceRet	Optional. A **Double** type variable.
minDistanceSet	Required. An expression that returns a **Double** value.

Remarks

The minimal distance between smart connectors and shapes is specified in internal ConceptDraw units (**InternalUnit**), and can be greater than or equal to 0. (any positive value). Any value of *minDistanceSet* less than **0** is equivalent to setting the distance to **0**. This property can also be changed from within ConceptDraw, using the menu "File->Document Properties->Advanced->Connectors And Routing".

See Also FlowAroundObjects property, LineJumpOrient property, LineJumpSize property, LineJumpType property, MaxNumberOfLegs property, PassThroughGroups property

Name Property

Name Property

A **String** type value. The name for the instance of an object in the **Applies to** list.

Applies to: DataSourceValue object, Document object, Layer object, Library object, Master object, Page object, ServObj object, Shape object, Style object

Syntax
[[**Let**] *nameRet* =] *object*.**Name**

[**Let**] *object*.**Name** = *nameSet*

The **Name** property syntax has these Elements:

Element	**Description**

object	Required. An expression that returns an object in the **Applies to** list.
nameRet	Optional. A **String** type variable.
nameSet	Required. An expression that returns a **String** value. The new filename of the document/library.

Remarks

Below is the meaning of the **Name** property for different objects:

Object	**Name property description**
Document	Gets or sets the name of the document under which it will be saved. Don't confuse **Name** with the FullName, Path or Title properties, as **Name** is used only for working with filenames which don't include the full path. If you change the **Name** property, the FullName property is changed as well (FullName represents concatenated **Name** and Path strings.)
Library	Gets and sets the file name of the library. Has the same meaning and function as the **Name** property of a document.
Master	The name (title) of a master object (library object).
Page	The name (title) of a document page.
ServObj	The name (title) of a service object.
Style	The name (title) of a style in the style collection of a document. It is unique within the scope of the style collection of the document.
Shape	The name (title) of a shape.
Layer	The name (title) of a layer.
DataSourceValue	Data from the **Name** field of the object **Data** parameters table .

Example

This example contains an application-level script. It demonstrates using the **Name** property of different objects.

```
TRACE thisDoc.Name
TRACE thisApp.Lib(1).Name
TRACE thisApp.Lib(1).Master(1).Name
TRACE thisPage.Name
TRACE thisPage.ServObj(1).Name
TRACE thisDoc.Style(1).Name
TRACE thisShape.Name
TRACE thisDoc.Layer(1).Name
TRACE thisShape.DSVALUE(1).Name
```

See Also FullName property, Path property, Title property

NonPrinting Property

NonPrinting Property

A **Boolean** type property. A flag that specifies whether to print this shape or not. **True** - the shape is non-printable, **False** - the shape is printable.

Applies to: Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**NonPrinting**

[**Let**] *object*.**NonPrinting** = *booleanSet*

The **NonPrinting** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
booleanRet	Optional. A **Boolean** type variable.
booleanSet	Required. An expression that returns a **Boolean** value.

Remarks

The **NonPrinting** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **NonPrinting** as a table parameter, use the **CDPT_NONPRINTING** constant tag.

See Also NonPrinting property, ResizeBehaviour property, ShowAlignBox property, ShowControlHandles property, ShowShapeHandles property, ShowText property

ObjType Property

Read-only. A **Long** type property, gets the type of the shape/service object.

Applies to: ServObj object, Shape object

Syntax

[[**Let**] *shapeTypeRet* =] *object*.**ObjType**

The **ObjType** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object from the **Applies to** list.
shapeTypeRet	Optional. A **Long** type variable.

Remarks

The **ObjType** property can take one of the following values:

Constant	Value	Description
cdUnknown	0	Unknown object type .
cdShape	1	A simple ConceptDraw shape.
cdGroup	2	A group.
cdConnector	3	A connector.
cdSmartConnector	4	A smart connector.
cdVectorPicture	5	A ConceptDraw Vector Picture.
cdRasterPicture	6	An object that contains raster image.
cdMFPicture	7	
cdPictPicture	8	
cdOLEObject	9	An OLE-object.
cdGuide	10	A guide line.

To check, whether the shape is a 1D-shape or a 2D-shape, use the **Is1D** property.

See Also

Is1D property

OnCmdArgs Property

OnCmdArgs Property

A **String** value. Gets or sets the arguments string of a menu item.

Applies to: MenuItem object

Syntax

[[**Let**] *argsRet* =] *object*.**OnCmdArgs**

[**Let**] *object*.**OnCmdArgs** = *argsSet*

The **OnCmdArgs** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an **MenuItem** object.
argsRet	Optional. A **String** type variable.
argsSet	Required. An expression that returns a **String** value.

OnCmdModule Property

OnCmdModule Property

A **String** value. Read-only. Returns the name of the external module with a processing procedure to process menu item command.

Applies to: MenuItem object

Syntax

[[**Let**] *moduleRet* =] *object*.**OnCmdModule**

The **OnCmdModule** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an **MenuItem** object.
moduleRet	Optional. A **String** type variable.

OnCmdSub Property

OnCmdSub Property

A **String** value. Read-only. Returns the name of the processing procedure to process menu item command.

Applies to: MenuItem object

Syntax

[[**Let**] *procRet* =] *object*.**OnCmdSub**

The **OnCmdSub** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an **MenuItem** object.
procRet	Optional. A **String** type variable.

PageID Property

PageID Property

Read-only. A **Long** value. An identifier (the ID property) of a page to which the hyperlink points.

Applies to: Hyperlink object

Syntax

[[**Let**] *pageIDRet* =] *object*.**PageID**

The **PageID** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Hyperlink** object.
pageIDRet	Optional. A **Long** type variable.

Remarks

The **PageID** property is effective if the hyperlink points to a page in a ConceptDraw document (the LinkType propery).

See Also LinkType property, ID property, ShapeID property

PageSizeX Property

PageSizeX Property

A **Double** type property. Gets and sets the width (the horizontal size) of a document page.

Applies to: Document object

Syntax

[[**Let**] *xSizeRet* =] *object*.**PageSizeX**

[**Let**] *object*.**PageSizeX** = *xSizeSet*

The **PageSizeX** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Document** object.
xSizeRet	Optional. A **Double** type variable.
xSizeSet	Required. An expression that returns a **Double** value. The new value for the document page width.

Remarks

All pages of the document have the same width. The value of the page width is set in the internal units of ConceptDraw (**InternalUnit**).

The **PageSizeX** property can take only positive values. Changing the value of **PageSizeX** re-draws all pages of the document to reflect the new width. The value of the **PageSizeX** property can be also viewed and changed in the following dialog in ConceptDraw: "File->Document Properties->Page".

Example

```
thisDoc.PageSizeX = 700
```

See Also PageSizeY property

PageSizeY Property

PageSizeY Property

A **Double** type property. Gets and sets the height (the vertical size) of a document page.

Applies to: Document object

Syntax

[[**Let**] *ySizeRet* =] *object*.**PageSizeY**

[**Let**] *object*.**PageSizeY** = *ySizeSet*

The **PageSizeY** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
ySizeRet	Optional. A **Double** type variable.
ySizeSet	Required. An expression that returns a **Double** value. The new value for the document page height.

Remarks

All pages of the document have the same height. The value of the page height is set in the internal units of ConceptDraw (**InternalUnit**).

The **PageSizeY** property can take only positive values. Changing the value of **PageSizeY** re-draws all pages of the document to reflect the new height. The value of the **PageSizeY** property can be also viewed and changed in the following dialog in ConceptDraw: "File->Document Properties->Page".

Example

```
thisDoc.PageSizeY = 700
```

See Also PageSizeX property

Page Property (ServObj, Shape objects)

Page Property (SerbObj, Shape objects)

Read-only. Gets a **Page** object corresponding to the page, which contains the object from the **Applies to** list.

Applies to: SerbObj object, Shape object

Syntax

[[**Set**] *pageRet* =] *object*.**Page**

The **Page** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
pageRet	Optional. A **Page** type variable.

Remarks

For shapes inside a library this property always returns **Nothing**.

See Also Document property, Parent property, Page object

Page Property (Window object)

Page Property (Window object)

Read-only. Gets a **Page** object for the page that is displayed in the window.

Applies to: Window object

Syntax

[[**Set**] *pageRet* =] *object*.**Page**

The **Page** property syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Window** object.
pageRet	Optional. A **Page** type variable.

Remarks

If the window represents a document window (document view, see the **Type** property) the **Page** property returns an instance of the **Page** object, corresponding to the page, displayed in the window. For windows of all other types the **Page** property returns **Nothing**. If the window is an Edit Group window, **Page** returns the page, to which the group belongs.

See Also Document property, Library property, Shape property, Page object

Paragraph Property

Paragraph Property

Read-only. Returns a **Paragraph** object that contains parameters of the paragraph's text for this style.

Applies to: Style object

Syntax

[[**Set**] *paragraphRet* =] *object*.**Paragraph**

The **Paragraph** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Style** object.

paragraphRet	Optional. A **Paragraph** type variable.

Remarks

You can't modify the instance of the **Paragraph** object, stored in the **Paragraph** property of the style. However, you can change parameters of this instance of the **Paragraph** object. When a style is assigned to a shape, the parameters of the **Paragraph** property of the style are set to all paragraphs of the shape's text. The **Paragraph** property is only effective when the **HasParaAttr** property of this style is **True**.

See Also	Character property, HasParaAttr property, TextBlock property, Paragraph object

Parent Property

Parent Property

Read-only. Returns an instance of the **Menu** object corresponding to the parent menu of the menu or menu item.

Applies to: Menu object, MenuItem object

Syntax

[[Set] *parentMenuRet* =] *object*.**Parent**

The **Enabled** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object in the **Applies to** list.
parentMenuRet	Optional. A **Menu** type variable.

Remarks

For a **MenuItem** object (a menu item) **Parent** returns the menu, which contains the given menu item. For a **Menu** object (a menu) **Parent** returns the menu which contains a menu item containing the given menu. For an application or document-level user-defined menu (see

CustomMenu property), the **Parent** property returns **Nothing** as an upper-level menu doesn't have a parent menu.

Parent Property (ServObj, Shape objects)

Parent Property (ServObj, Shape objects)

Read-only. Returns a **Shape** object that corresponds to the group (parent group) which owns this shape / service object.

Applies to: ServObj object, Shape object

Syntax

[[**Set**] *parentShapeRet* =] *object*.**Parent**

The **Parent** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
parentShapeRet	Optional. A **Shape** type variable.

Remarks

If *object* is located on a document page, the **Parent** property returns **Nothing** as in this case the parent object for this object is the page (the **Page** property). For shapes inside a library this property always returns **Nothing**.

See Also Document property, Page property, Page object

PassThroughGroups Property

PassThroughGroups Property

A **Boolean** property. Gets or sets a flag, that specifies whether the smart connectors in the document flow around the entire group (the **False** value), or pass through the group, flowing around the shapes inside it (the **True** value). Applies to all smart connectors of the document.

Applies to: Document object

Syntax

[[**Let**] *passThroughRet* =] *object*.**PassThroughGroups**

[**Let**] *object*.**PassThroughGroups** = *passThroughSet*

The **PassThroughGroups** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Document** object.
passThroughRet	Optional. A **Boolean** type variable.
passThroughSet	Required. An expression that returns a **Boolean** value.

Remarks

The **PassThroughGroups** property can also be viewed or modified by using the dialog in ConceptDraw: "File->Document Properties->Advanced->Connectors And Routing".

See Also	FlowAroundObjects property, LineJumpOrient property, LineJumpSize property, LineJumpType property, MaxNumberOfLegs property, MinDistToShapes property

Path Property

Path Property

A **String** value. Gets or sets the path to the file for an object in the **Applies to** list.

Applies to: Document object, Library object

Syntax

[[**Let**] *pathRet* =] *object*.**Path**

[**Let**] *object*.**Path** = *pathSet*

The **Path** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
pathRet	Optional. A **String** type variable.
pathSet	Required. An expression that returns a **String** value. The new path to the document/library file.

Remarks

This property is modified automatically when the document or library are saved with a different path. The **Path** property contains only the path to the file, without the filename. Use the **Name** property to find out the name of the document/library. The full name with path is contained in the **FullName** property.

See Also

FullName property, Name property

PenColor Property

PenColor Property

Read-only. Returns an instance of the **Color** object, that contains information about the line color of the shape.

Applies to: Shape object, Style object

Syntax

[[**Set**] *colorRet* =] *object*.**PenColor**

The **PenColor** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
colorRet	Optional. A **Color** type variable.

Remarks

Shape object:
The **PenColor** property is also a table parameter of the shape, that is, its value can be described by a formula. The instance of the **Color** object, that contains the **PenColor** property is not changed, instead, the color components contained in **PenColor** are changed. To work with the **PenColor** property as with a table parameter, use the constant tag **CDPT_LINECOLOR**.

Style object:
When a style is assigned to a shape, the parameters of the **PenColor** property of the style are set to the **PenColor** property of the shape. **PenColor** is only effective when the **HasPenAttr** property of this style is **True**.

See Also	DefPenColor property, HasPenAttr property, PenPattern property, PenWeight property, Color object

PenPattern Property

PenPattern Property

A **Long** type property. The line pattern type for the shape.

Applies to: Shape object, Style object

Syntax

[[**Let**] *linePatternRet* =] *object*.**PenPattern**

[**Let**] *object*.**PenPattern** = *linePatternSet*

The **PenPattern** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
linePatternRet	Optional. A **Long** type variable.
linePatternSet	Required. An expression that returns a **Long** value.

Remarks

The value of **PenPattern** can be in the range of 0 to 15.

Shape object:
The **PenPattern** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **PenPattern** as a table parameter, use the **CDPT_PENPATTERN** constant tag.

Style object:
When a style is assigned to a shape, the parameters of the **PenPattern** property of the style are set to the **PenPattern** property of the shape. **PenPattern** is only effective when the **HasPenAttr** property of this style is **True**.

See Also DefPenPattern property, HasPenAttr property, PenColor property, PenWeight property

PenWeight Property

PenWeight Property

A **Long** type property. The width of the shape's lines.

Applies to: Shape object, Style object

Syntax

[[**Let**] *longRet* =] *object*.**PenWeight**

[**Let**] *object*.**PenWeight** = *penWeightSet*

The **PenWeight** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
longRet	Optional. A **Long** type variable.
penWeightSet	Required. An expression that returns a **Long** value.

Remarks

Line weight is set in **points** (1 pt = 1/72 inch).

Shape object:
The **PenWeight** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **PenWeight** as a table parameter, use the **CDPT_LINEWEIGHT** constant tag.

Style object:
When a style is assigned to a shape, the parameters of the **PenWeight** property of the style are set to the **PenWeight** property of the shape. **PenWeight** is only effective when the **HasPenAttr** property of this style is **True**.

See Also	DefPenWeight property, HasPenAttr property, PenPattern property, PenColor property

Pos Property (Character object)

Pos Property (Character object)

A **Byte** type property. The position with respect to the text baseline (subscript, superscript) of this character block.

Applies to: Character object

Syntax

[[**Let**] *byteRet* =] *object*.**Pos**

[**Let**] *object*.**Pos** = *posSet*

The **Pos** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Character** object.
byteRet	Optional. A **Byte** type variable.
posSet	Required. An expression that returns a **Byte** value.

Remarks

The **Pos** property can take one of the following values:

Constant	**Value**	**Description**

cdPosNormal	0	Normal text size and position.
cdPosSuper	1	Superscript.
cdPosSub	2	Subscript.

The **Pos** property is also a table parameter of the shape which contains the *object* character block, that is, its value can be described by a formula. To work with **Pos** as a table parameter, use the **CDPT_CHAR_POS** constant tag.

Example

```
Dim MyShape As Shape
' Assume Shape with ID 1 is on the current page of the document
Set MyShape = thisDoc.ActivePage.ShapeByID(1)
' Change position of MyShape.Character(2) to subscript
' (assume such shape exists)
MyShape.Character(2).Pos = cdbPosSub
' Inform ConceptDraw Engine about the changes.
MyShape.PropertyChanged(CDPT_CHAR_POS)
```

See Also SetCharPos method

Pos Property (TabStop object)

Pos Property (TabStop object)

A **Single** type property. Specifies the interval between this tab stop and the left edge of the text block which contains this tab stop.

Applies to: TabStop object

Syntax

[[**Let**] *byteRet* =] *object*.**Pos**

[**Let**] *object*.**Pos** = *posSet*

The **Pos** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **TabStop** object.
byteRet	Optional. A **Byte** type variable.

posSet	Required. An expression that returns a **Byte** value.

Remarks

The distance between the tab stop and the left edge of the text block is measured in internal ConceptDraw units (**InternalUnit**).

The **Pos** property is also a table parameter of the shape which contains the text block including the *object* tab stop, that is, its value can be described by a formula. To work with **Pos** as a table parameter, use the **CDPT_TABPOS** constant tag.

See Also	Align property, Shape object, TextBlock object

Printable Property

Printable Property

A Boolean value. Specifies whether the layer is printable.

Applies to objects: Layer

Syntax

[[**Let**] *RetVal* =] *object*.**Printable**

[**Let**] *object*.**Printable** = *SetVal*

The **Printable** property syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
RetVal	A Boolean type variable.
SetVal	A Boolean value.

Remarks

If **Printable** property is FALSE, the shapes on the layer won't be printed. You can set the **Printable** property to FALSE only when the layer is not active.

Example

```
Dim MyLayer as Layer
' Get second Layer of  thisDoc
set MyLayer = thisDoc.Layer(2)
' Make it non-printable
' (assume MyLayer is not active layer)
MyLayer.Printable = FALSE
```

See Also Layer Object, Document Object

Prompt Property

Prompt Property

A **String** type property. Represents the prompt for an object from the **Applies to** list.

Applies to: Action object, CustomProp object, Menu object, MenuItem object

Syntax

[[**Let**] *promptRet* =] *object*.**Prompt**

[**Let**] *object*.**Prompt** = *promptSet*

The **Prompt** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
promptRet	Optional. A **String** type variable.
promptSet	Required. An expression that returns a **String** value.

Remarks

For the **Menu** and **MenuItem** objects the **Prompt** property specifies the prompt that appears when you position the pointer over the menu or the menu item.

Example

```
Dim s as Shape, MyAction as Action
' Assume there is a shape with ID 1 on the active page.
' Assume the shape contains at least one action
s = thisDoc.ActivePage.ShapeByID(1)
' Get the reference to an Action object
Set MyAction = s.Action(1)
```

```
' Set the prompt for the action in the menu
MyAction.Prompt = "Hide Shape"
```

See Also Desc property, Name property, Title property

Red Property

Red Property

Gets or sets an **Integer** value, that represents the red component of an RGB color.

Applies to: Color object, ColorEntry object

Syntax

[[**Let**] *redRet* =] *object*.**Red**

[**Let**] *object*.**Red** = *redSet*

The **Red** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object in the **Applies to** list.
redRet	Optional. An **Integer** type variable.
redSet	Required. An expression that returns an **Integer** value.

Remarks

The **Red** property is only effective if the color is an RGB color (see the **IsRGB** property).

Example

This example contains a document-level script. It demonstrates how to find out the value of the red component of the fill color (in RGB format) of a Shape object.

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
If s.FillColor.IsRGB <> false  Then    ' An RGB color ?
 MsgBox(s.FillColor.Red)    '  If yes, display the value of the red component.
endif
```

See Also Blue property, Green property, Red property, IsRGB Property

Refresh Property

Refresh Property

A **Long** type property. Time interval in seconds through which occurs updatings of data from a source. By default **Refresh** property is equal 1 second.

Applies to: DataSource object

Syntax

[[**Let**] *RefreshRet* =] *object*.**Refresh**

[**Let**] *object*.**Refresh** = *RefreshSet*

The **Refresh** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **DataSource** object.
RefreshRet	Optional. A **Long** type variable.
RefreshSet	Required. An expression that returns a **Long** value.

Remarks

The **Refresh** property is also a table parameter of the DataSource, that is, its value can be described by a formula. To work with **Refresh** as a table parameter, use the **CDPT_DS_REFRESH_TIME** constant tag.

Example

```
dim ds as DATASOURCE
ds = thisShape.DATASOURCE(1)
trace ds.Refresh
ds.Refresh = 45
trace ds.Refresh
or
thisShape.SetPropertyFormula("25", CDPT_DS_REFRESH_TIME, 1)
trace ds.Refresh
```

See Also	DataSource object, Action property, Active property, DataSource property, ShowErrors property, ShowWarnings property, Timeout property

ResizeBehaviour Property

ResizeBehaviour Property

A **Byte** type property. Determines how the shape behaves when its parent group is being resized.

Applies to: Shape object

Syntax

[[**Let**] *rbehRet* =] *object*.**ResizeBehaviour**

[**Let**] *object*.**ResizeBehaviour** = *rbehSet*

The **ResizeBehaviour** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
rbehRet	Optional. A **Byte** type variable.
rbehSet	Required. An expression that returns a **Byte** value.

Remarks

The **ResizeBehaviour** property can take the following values:

Constant	**Value**	**Description**
cdRBScaleWithGroup	0	The shape moves and changes its dimensions together with its parent group.
cdRBRepositionOnly	1	The shape moves together with the parent group, but its size doesn't change.
cdRBUseGroupSettings	2	The shape behaves according to the Resize Behavior settings of the parent group.

The **ResizeBehaviour** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **ResizeBehaviour** as a table parameter, use the **CDPT_RESIZEBEHAVIOUR** constant tag.

See Also NonPrinting property, ShowAlignBox property, ShowControlHandles property, ShowShapeHandles property, ShowText property

RightInd Property

RightInd Property

A **Singe** type property. The distance all lines of text in a paragraph are indented from the right margin of the text block.

Applies to: Paragraph object

Syntax

[**Let**] *singleRet* = *object*.**RightInd**

[**Let**] *object*.**RightInd** = *rightIndSet*

The **RightInd** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Paragraph** object.
singleRet	Optional. A **Single** type variable.
rightIndSet	Required. An expression that returns a **Single** value.

Remarks

Indents are specified in internal ConceptDraw units (**InternalUnit**).

The **RightInd** property is also a table parameter of the shape that contains the *object* paragraph, that is, its value can be described by a formula. To work with **RightInd** as a table parameter, use the **CDPT_PARA_RIGHTIND** constant tag.

Example

This example demonstrates how to set a right indent for the second paragraph of a shape. It assumes there is a shape on the current page, and its text contains at least two paragraphs.

```
Dim s as Shape
```

```
s = thisDoc.ActivePage.ShapeByID(1)
' Move the second paragraph of by 100 points left from the right border of the
text block.
s.Paragraph(2).RightInd = 100
' Inform ConceptDraw Engine about the changes for re-drawing.
s.PropertyChanged(CDPT_PARA_RIGHTIND)
```

See Also SetParaRightInd method

RightMargin Property

RightMargin Property

A **Single** type property. The distance the text inside the text block is offset from the right border of the text box.

Applies to: TextBlock object

Syntax

[[**Let**] *sinleRet* =] *object*.**RightMargin**

[**Let**] *object*.**RightMargin** = *rightMarginSet*

The **RightMargin** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **TextBlock** object.
sinleRet	Optional. A **Single** type variable.
rightMarginSet	Required. An expression that returns a **Single** value.

Remarks

The unit of measure for the **RightMargin** property are internal ConceptDraw units (**InternalUnit**).

The **RightMargin** property is also a table parameter of the shape which contains the *object* text block, that is, its value can be described by a formula. To work with **RightMargin** as a table parameter, use the **CDPT_RIGHTMARGIN** constant tag.

Example

This example demonstrates how to increase the distance between the text and right border of the text block of an existing shape.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Increase the distance between the text and the right border of the text box
by 20 points.
s.TextBlock.RightMargin = 20
' Inform ConceptDraw Engine about the changes for re-drawing.
s.PropertyChanged(CDPT_RIGHTMARGIN)
```

Right Property

Right Property

Gets or sets a Double value, representing the coordinate of the rightmost point of an instance of the object.

Applies to objects: DRect

Syntax

[**Let**] *RetVal* = *object*.**Right**

[**Let**] *object*.**Right** = *SetVal*

Element	Description
object	A reference to an instance of the object.
RetVal	A Double type variable.
SetVal	A Double value.

Example

```
Dim MyObject as new DRect   ' Create an instance of the object
MyObject.Right = 200
```

See Also DRect Object

RoundCorners Property

RoundCorners Property

A **Double** type property. The corner radius of the shape.

Applies to: Shape object

Syntax

[[**Let**] *doubleRet* =] *object*.**RoundCorners**

[**Let**] *object*.**RoundCorners** = *roundCornerSet*

The **RoundCorners** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
doubleRet	Optional. A **Double** type variable.
roundCornerSet	Required. An expression that returns a **Double** value.

Remarks

The **RoundCorners** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **RoundCorners** as a table parameter, use the **CDPT_ROUNDCORNERS** constant tag.

Scale Property

Scale Property

A **Double** type property. Gets or sets the drawing scale in the document. The drawing scale is the ratio of the dimensions in the drawing to the actual size of the objects represented by shapes in a ConceptDraw drawing.

Applies to: Document object

Syntax

[[**Let**] *scaleRet* =] *object*.**Scale**

[**Let**] *object*.**Scale** = *scaleSet*

The **Scale** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
scaleRet	Optional. A **Double** type variable.
scaleSet	Required. An expression that returns a **Double** value.

Remarks

The **Scale** property can only have positive values, because a drawing scale can't be negative or zero. You can also modify **Scale** from within ConceptDraw, using the "File->Document Properties->Settings" dialog. Note, that modifying the **Scale** property sets the document scale "Custom Scale" and automatically re-calculates the two components of the document scale: the ConceptDraw units of measure and the real units of measure of the shapes in the document.

Example

This example contains an application-level script. It draws a rectangle, which has a formula associated with **Scale** property of the document. Then **Scale** is changed from 0.001 to 1.000. The changes are immediately reflected in the shape.

```
' Declare variables
Dim shp As Shape
' Draw a rectangle
Set shp = thisDoc.ActivePage.DrawRect( 100,100,1000,500 )
' Assign text to rectangle
shp.Text = " "
' Set font size for rectangle's text
shp.SetCharSize(1,1,22)
' Set the DocScale formula
' for the Text property of the rectangle
shp.SetPropertyFormula( "DocScale", CDPT_TEXT )
MsgBox("Let's Start!")
' Change the document scale from 0.001 to 1.000
for i=0.001 to 1.001 Step 0.001
    thisDoc.Scale = i
next i
```

See Also SnapSensitivity property, SplineSmooth property

ShadowColor Property

ShadowColor Property

Read-only. Returns an instance of the **Color** object that corresponds to the shadow color of the shape.

Applies to: Shape object, Style object

Syntax

[[**Set**] *colorRet* =] *object*.**ShadowColor**

The **ShadowColor** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
colorRet	Optional. A **Color** type variable.

Remarks

Shape object:
The **ShadowColor** property is also a table parameter of the shape, that is, its value can be described by a formula. The instance of the **Color** object, that contains the **ShadowColor** property is not changed, instead, the color components contained in **ShadowColor** are changed. To work with the **ShadowColor** property as with a table parameter, use the constant tag **CDPT_SHADOWCOLOR**.

Style object:
When a style is assigned to a shape, the parameters of the **ShadowColor** property of the style are set to the **ShadowColor** property of the shape. **ShadowColor** is only effective when the **HasShadowAttr** property of this style is **True**.

See Also	DefShadowColor property, HasShadowAttr property, ShadowPatColor property, ShadowPattern property, Color object

ShadowOffsetX Property

ShadowOffsetX Property

A **Double** type property. Gets or sets the horizontal shadow offset for the shapes of the document. Applies to all shapes in the document, that have shadow.

Applies to: Document object

Syntax

[[**Let**] *xOffsetRet* =] *object*.**ShadowOffsetX**

[**Let**] *object*.**ShadowOffsetX** = *xOffsetSet*

The **ShadowOffsetX** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
xOffsetRet	Optional. A **Double** type variable.
xOffsetSet	Required. An expression that returns a **Double** value. The new value for the horizontal shadow offset.

Remarks

The shadow offset is specified in the internal units of ConceptDraw (**InternalUnit**). The range of valid values is not limited. When **ShadowOffsetX** is modified, all shapes in the document that have shadow are redrawn to reflect the new value. You can also modify the **ShadowOffsetX** property from within ConceptDraw in the "File->Document Properties->Settings" dialog.

See Also ShadowOffsetY property

ShadowOffsetY Property

ShadowOffsetY Property

A **Double** type property. Gets or sets the vertical shadow offset for the shapes of the document. Applies to all shapes in the document, that have shadow.

Applies to: Document object

Syntax

[[**Let**] *yOffsetRet* =] *object*.**ShadowOffsetY**

[**Let**] *object*.**ShadowOffsetY** = *yOffsetSet*

The **ShadowOffsetY** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
yOffsetRet	Optional. A **Double** type variable.
yOffsetSet	Required. An expression that returns a **Double** value. The new value for the vertical shadow offset.

Remarks

The shadow offset is specified in the internal units of ConceptDraw (**InternalUnit**). The range of valid values is not limited. When **ShadowOffsetY** is modified, all shapes in the document that have shadow are redrawn to reflect the new value. You can also modify the **ShadowOffsetY** property from within ConceptDraw in the "File->Document Properties->Settings" dialog.

See Also ShadowOffsetX property

ShadowPatColor Property

ShadowPatColor Property

Read-only. Returns an instance of the **Color** object that corresponds to the shadow pattern color of the shape.

Applies to: Shape object, Style object

Syntax

[[**Set**] *colorRet* =] *object*.**ShadowPatColor**

The **ShadowPatColor** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
colorRet	Optional. A **Color** type variable.

Remarks

Shape object:
The **ShadowPatColor** property is also a table parameter of the shape, that is, its value can be described by a formula. The instance of the **Color** object, that contains the **ShadowPatColor** property is not changed, instead, the color components contained in **ShadowPatColor** are changed. To work with the **ShadowPatColor** property as with a table parameter, use the constant tag **CDPT_SHADOWPATCOLOR**.

Style object:
When a style is assigned to a shape, the parameters of the **ShadowPatColor** property of the style are set to the **ShadowPatColor** property of the shape. **ShadowPatColor** is only effective when the **HasShadowAttr** property of this style is **True**.

See Also	DefShadowPatColor property, HasShadowAttr property, ShadowColor property, ShadowPattern property, Color object

ShadowPattern Property

ShadowPattern Property

A **Long** type property. Gets and sets the shadow pattern of the shape.

Applies to: Shape object, Style object

Syntax

[[**Let**] *longRet* =] *object*.**ShadowPattern**

[**Let**] *object*.**ShadowPattern** = *patternSet*

The **ShadowPattern** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
longRet	Optional. A **Long** type variable.
patternSet	Required. An expression that returns a **Long** value.

Remarks

Shape object:
The **ShadowPattern** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **ShadowPattern** as a table parameter, use the **CDPT_SHADOWPATTERN** constant tag.

Style object:
When a style is assigned to a shape, the parameters of the **ShadowPattern** property of the style are set to the **ShadowPattern** property of the shape. **ShadowPattern** is only effective when the **HasShadowAttr** property of this style is **True**.

See Also DefShadowPattern property, HasShadowAttr property, ShadowColor property, ShadowPatColor property

ShapeID Property

ShapeID Property

Read-only. A **Long** value. The indentifier (the ID property) of the shape, to which the hyperlink points.

Applies to: Hyperlink object

Syntax
[[**Let**] *shapeIDRet* =] *object*.**ShapeID**

The **ShapeID** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Hyperlink** object.
shapeIDRet	Optional. A **Long** type variable.

Remarks

The **ShapeID** property is effective if the hyperlink points to an object inside the ConceptDraw document (the LinkType property).

See Also LinkType property, ID property, PageID property

Shape Property

Shape Property

Read-only. Returns an instance of the **Shape** object, that represents a shape, associated with an instance of an object from the **Applies to** list.

Applies to: Master object, Window object

Syntax

[[**Set**] *shapeRet* =] *object*.**Shape**

The **Shape** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
shapeRet	Optional. A **Shape** type variable.

Remarks

Master object:
Returns the shape, contained in the library object (master object).

Window object:
If the window is a document window (document view, see the **Type** property) and is a Group Edit window at the same time, the **Shape** property returns an instance of the **Shape** object, that corresponds to the group displayed in the window. In all other cases the **Shape** property returns **Nothing**.

See Also Document property, Library property, Page property, Shape object

ShowAlignBox Property

A **Boolean** type property. A flag that specifies whether to show the shape's alignment box. **True** - the alignment box is visible. **False** - the alignment box is not visible.

Applies to: Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**ShowAlignBox**

[**Let**] *object*.**ShowAlignBox** = *booleanSet*

The **ShowAlignBox** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
booleanRet	Optional. A **Boolean** type variable.
booleanSet	Required. An expression that returns a **Boolean** value.

Remarks

The **ShowAlignBox** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **ShowAlignBox** as a table parameter, use the **CDPT_SHOWALIGNBOX** constant tag.

See Also	NonPrinting property, ResizeBehaviour property, ShowControlHandles property, ShowShapeHandles property, ShowText property

ShowControlHandles Property

A **Boolean** type property. A flag that specifies whether to show the shape's control handles. **True** - the control handles are visible. **False** - the control handles are not visible.

Applies to: Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**ShowControlHandles**

[**Let**] *object*.**ShowControlHandles** = *booleanSet*

The **ShowControlHandles** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
booleanRet	Optional. A **Boolean** type variable.
booleanSet	Required. An expression that returns a **Boolean** value.

Remarks

The **ShowControlHandles** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **ShowControlHandles** as a table parameter, use the **CDPT_SHOWCONTROLHANDLES** constant tag.

See Also NonPrinting property, ResizeBehaviour property, ShowAlignBox property, ShowShapeHandles property, ShowText property

ShowErrors Property

ShowErrors Property

A **Boolean** type property. Defines, whether it is necessary to display the corresponding icon at mistake emergence in the course of work with a source of data. **True** - inclusion of display of an icon. **False** - shutdown of display of an icon. By default **ShowErrors** property is equal to True.

Applies to: DataSource object

Syntax

[[**Let**] *ShowErrorsRet* =] *object*.**ShowErrors**

[**Let**] *object*.**ShowErrors** = *ShowErrorsSet*

The **ShowErrors** property syntax has these Elements:

Element	**Description**

object	Required. An expression that returns a **DataSource** object.
ShowErrorsRet	Optional. A **Boolean** type variable.
ShowErrorsSet	Required. An expression that returns a **Boolean** value.

Remarks

The **ShowErrors** property is also a table parameter of the DataSource, that is, its value can be described by a formula. To work with **ShowErrors** as a table parameter, use the **CDPT_DS_SHOW_ERRORS** constant tag.

Example

```
dim ds as DATASOURCE
ds = thisShape.DATASOURCE(1)
trace ds.ShowErrors
ds.ShowErrors = False
trace ds.ShowErrors
or
thisShape.SetPropertyFormula("True",CDPT_DS_SHOW_ERRORS, 1)
trace ds.ShowErrors
```

See Also	DataSource object, Action property, Active property, DataSource property, Refresh property, ShowWarnings property, Timeout property

ShowShapeHandles Property

ShowShapeHandles Property

A **Boolean** type property. A flag that specifies whether to show the shape's resize and rotation handles. **True** - the handles are visible. **False** - the handles are not visible.

Applies to: Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**ShowShapeHandles**

[**Let**] *object*.**ShowShapeHandles** = *booleanSet*

The **ShowShapeHandles** property syntax has these Elements:

Element	**Description**

object	Required. An expression that returns a **Shape** object.
booleanRet	Optional. A **Boolean** type variable.
booleanSet	Required. An expression that returns a **Boolean** value.

Remarks

The **ShowShapeHandles** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **ShowShapeHandles** as a table parameter, use the **CDPT_SHOWSHAPEHANDLES** constant tag.

See Also NonPrinting property, ResizeBehaviour property, ShowAlignBox property, ShowControlHandles property, ShowText property

ShowText Property

ShowText Property

A **Boolean** type property. A flag that specifies whether to show the shape's text. **True** - the text is visible. **False** - the text is not visible.

Applies to: Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**ShowText**

[**Let**] *object*.**ShowText** = *showTextSet*

The **ShowText** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
booleanRet	Optional. A **Boolean** type variable.
showTextSet	Required. An expression that returns a **Boolean** value.

Remarks

The **ShowText** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **ShowText** as a table parameter, use the **CDPT_SHOWTEXT** constant tag.

See Also	Text property, NonPrinting property, ResizeBehaviour property, ShowAlignBox property, ShowControlHandles property, ShowShapeHandles property

ShowWarnings Property

ShowWarnings Property

A **Boolean** type property. Defines, whether it is necessary to display the corresponding icon at emergence of remarks in the course of work with a source of data. **True** - inclusion of display of an icon. **False** - shutdown of display of an icon. By default **ShowWarnings** property is equal to True.

Applies to: DataSource object

Syntax

[[**Let**] *ShowWarningsRet* =] *object*.**ShowWarnings**

[**Let**] *object*.**ShowWarnings** = *ShowWarningsSet*

The **ShowWarnings** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **DataSource** object.
ShowWarnings Ret	Optional. A **Boolean** type variable.
ShowWarnings Set	Required. An expression that returns a **Boolean** value.

Remarks

The **ShowWarnings** property is also a table parameter of the DataSource, that is, its value can be described by a formula. To work with **ShowWarnings** as a table parameter, use the **CDPT_DS_SHOW_WARNINGS** constant tag.

Example

```
dim ds as DATASOURCE
ds = thisShape.DATASOURCE(1)
trace ds.ShowWarnings
ds.ShowWarnings = False
trace ds.ShowWarnings
or
thisShape.SetPropertyFormula("False",CDPT_DS_SHOW_WARNINGS, 1)
trace ds.ShowWarnings
```

See Also	DataSource object, Action property, Active property, DataSource property, Refresh property, ShowErrors property, Timeout property

Size Property

Size Property

An **Integer** type property. The font size of the character block.

Applies to: Character object

Syntax

[[**Let**] *integerRet* =] *object*.**Size**

[**Let**] *object*.**Size** = *sizeSet*

The **Size** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Character** object.
integerRet	Optional. A **Integer** type variable.
sizeSet	Required. An expression that returns a **Integer** value.

Remarks

The **Size** property cannot take negative values. The font size is specified in **points** (1 pt = 1/72 inch).

The **Size** property is also a table parameter of the shape which contains the *object* character block, that is, its value can be described by a formula. To work with **Size** as a table parameter, use the **CDPT_CHAR_SIZE** constant tag.

See Also SetCharSize method

SnapSensitive Property

SnapSensitivity Property

An **Integer** type property. Gets or sets the snap sensitivity value for a document.

Applies to: Document object

Syntax

[[**Let**] *snapSensRet* =] *object*.**SnapSensitivity**

[**Let**] *object*.**SnapSensitivity** = *snapSensSet*

The **SnapSensitivity** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
snapSensRet	Optional. A **Integer** type variable.
snapSensSet	Required. An expression that returns a **Integer** value.

Remarks

Snap sensitivity is set in screen pixels. The **SnapSensitivity** property can only have positive or zero values. An attempt to assign a negative value won't modify **SnapSensitivity.** You can also change snap sensitivity for a document from within ConceptDraw using the "File->Document Properties->Settings" dialog.

Example

This example contains an application-level script. It increases the snap sensitivity value for all open documents by 20%.

```
' Declare variable
```

```
Dim cur_doc As Document
' Get the first document
Set cur_doc = thisApp.FirstDoc()
While cur_doc <> Null
 ' Set snap sensitivity
 cur_doc.SnapSensitivity = cur_doc.SnapSensitivity * 1.2
 ' Get next document
 Set cur_doc = thisApp.NextDoc()
Wend
```

See Also Scale property, SplineSmooth property

Spacing Property

Spacing Property

A **Single** type property. The spacing between characters for this character block.

Applies to: Character object

Syntax

[[**Let**] *singleRet* =] *object*.**Spacing**

[**Let**] *object*.**Spacing** = *spacingSet*

The **Spacing** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Character** object.
singleRet	Optional. A **Single** type variable.
spacingSet	Required. An expression that returns a **Single** value.

Remarks

The **Spacing** property can take any positive or zero values.

The **Spacing** property is also a table parameter of the shape which contains the *object* character block, that is, its value can be described by a formula. To work with **Spacing** as a table parameter, use the **CDPT_CHAR_SPACING** constant tag.

Example

```
Dim MyShape As Shape
' Shape with ID 1 must exist on the current page
Set MyShape = thisDoc.ActivePage.ShapeByID(1)
' Change character spacing for MyShape.Character(2)
' (assume Character(2) exists in MyShape)
MyShape.Character(2).Pos = 20  'change spacing property
'  Inform ConceptDraw Engine about the changes
MyShape.PropertyChanged(CDPT_CHAR_SPACING)
```

See Also SetCharSpacing method

SplineSmooth Property

SplineSmooth Property

A **Long** type property. Gets and sets the spline smoothness value for shapes of the document. It's set as the default value for new shapes, that contain spline segments.

Applies to: Document object

Syntax

[[**Let**] *splineSmoothRet* =] *object*.**SplineSmooth**

[**Let**] *object*.**SplineSmooth** = *splineSmoothSet*

The **SplineSmooth** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns a **Document** object.
snapSensRet	Optional. A **Long** type variable.
snapSensSet	Required. An expression that returns a **Long** value.

Remarks

The spline smoothness value is specified in percent, that is, the value of the **SplineSmooth** property may range from **0** to **100**. An attempt to set any other value is ignored.

See Also Scale property, SnapSensitivity property, SplineSmooth property

State Property

State Property

Read-only. A **Long** type property. Returns the state of the window.

Applies to: Window object

Syntax

[[**Let**] *stateRet* =] *object*.**State**

The **State** property syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns a **Window** object.
stateRet	Optional. A **Long** type variable.

Remarks

The **State** property can take the following values:

Value	**Constant**	**Description**
0	cdNormal	This window has the normal position and size.
1	cdMaximized	The window is maximized to full screen.
2	cdMinimized	The window is minimized.

The state of the window and the value of the **State** property can be modified by using the following methods: **Maximize**, **Minimize** and **Restore**.

Example

This example contains a document-level script. It demonstrates how the value of the **State** property is changed by using the **Maximize**, **Minimize** and **Restore** methods.

```
' Declare variables
Dim state As Integer
'  Remember the state of the document window
state = thisDoc.FirstView().State
```

```
' Display the current state
TRACE state
' Maximize the window
thisDoc.FirstView().Maximize()
MsgBox( "State = " & thisDoc.FirstView().State )
' Minimize the  window
thisDoc.FirstView().Minimize()
MsgBox( "State = " & thisDoc.FirstView().State )
' Set the normal state of the window
thisDoc.FirstView().Restore()
MsgBox( "State = " & thisDoc.FirstView().State )
' Restore the original state of the window
If state = 0 Then
    thisDoc.FirstView().Restore()
Else If state = 1 Then
    thisDoc.FirstView().Maximize()
Else If state = 2 Then
    thisDoc.FirstView().Minimize()
End If
```

See Also Type property, Maximize method, Minimize method, Restore method

Style Property

Style Property

A **Byte** type property. Specifies the font style (bold, italic, underline, etc.) for this character block.

Applies to: Character object

Syntax

[[**Let**] *byteRet* =] *object*.**Style**

[**Let**] *object*.**Style** = *styleSet*

The **Style** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Character** object.
byteRet	Optional. A **Byte** type variable.
styleSet	Required. An expression that returns a **Byte** value.

Remarks

The **Style** property can take any combination of the values listed below:

Constant	**Value**	**Description**
cdFSNormal	0	Normal.
cdFSBold	1	Bold.
cdFSItalic	2	Italic.
cdFSUnderline	4	Underline.
cdFSStrikeTrough	8	Strikethrough.

The **Style** property is also a table parameter of the shape which contains the *object* character block, that is, its value can be described by a formula. To work with **Style** as a table parameter, use the **CDPT_CHAR_STYLE** constant tag.

See Also SetCharStyle method

SubID Property

SubID Property

Read-only. A **Long** type property. Returns the sub ID of the shape/service object, which is an unique integer number within the scope of the shape's parent object (group or page).

Applies to: Shape object, ServObj object

Syntax

[[**Let**] *subIDRet* =] *object*.**SubID**

The **SubID** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
subIDRet	Optional. A **Long** type variable.

Remarks

Note that the **SubID** property of the object is unique only within the scope of the object's parent object. That is, the objects in different groups or on different pages can have the same **SubID**s. In order to identify an object within one document, use the **ID** property.

See Also ID property, Page property, Parent property

Subj Property

Subj Property

A **String** value. Gets or sets a string that contains a brief description (subject) of a document/library.

Applies to: Document object, Library object

Syntax

[[**Let**] *subjRet* =] *object*.**Subj**

[**Let**] *object*.**Subj** = *subjSet*

The **Subj** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
subjRet	Optional. A **String** type variable.
subjSet	Required. An expression that returns a **String** value.

Remarks

The **Subj** property contains an empty string for any new document or library. The **Subj** property can also be changed in the dialogs in ConceptDraw: "File->Document Properties->General" for a document, "Library->Library Properties" - for a library.

Example

This example contains an application-level script. It closes all open documents for which the **Subj** property contains an empty string.

```
' Loop that goes through all open documents
' starting from the end.
For i=thisApp.DocsNum() To 1 Step -1
 ' If the document has no subject, close it.
 if thisApp.Doc(i).Subj = "" OR thisApp.Doc(i).Subj = Null Then
  thisApp.CloseDoc( thisApp.Doc(i) )
 End If
Next i
```

See Also Title property, Author property, Company property, Desc property

SubMenu Property

SubMenu Property

Read-only. Returns the submenu of the specified menu item.

Applies to: MenuItem object

Syntax

[[**Set**] *subMenuRet* =] *object*.**SubMenu**

The **SubMenu** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **MenuItem** object.
subMenuRet	Optional. A **Boolean** type variable.

Remarks

This property is only effective for the menu items of the **cdMenuItemPopup** type (the **Type** property), that is, those containing a submenu. For menu items of other types this property returns **Nothing**.

See Also Type property (MenuItem object), Menu object

TemplatesPath Property

TemplatesPath Property

Read-only. A **String** value. Returns the full way to files which are on the way, adjusted in **Preferences** appendix dialogue in the **Paths** tab in the field of **Templates**.

Applies to: Application object

Syntax

[[**Let**] *TemplatesPathRet* =] *object*.**TemplatesPath**

The **TemplatesPath** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Application** object.
TemplatesPathRet	Optional. A **String** type variable.

Remarks

The **TemplatesPath** property by default matters: **"personal folder of the user /AppData/Local/CS Odessa/ConceptDraw Office/ConceptDraw DIAGRAM DIAGRAM/Templates"**.

See Also Application object, DocumentsPath property, HelpPath property, LibrariesPath property

TextAngle Property

TextAngle Property

A **Double** type property. Represents the angle to which the shape's text box is rotated counterclockwise around the shape's rotation center. The angle is measured with respect to the horizontal (X) axis.

Applies to: Shape object

Syntax

[[**Let**] *textAngleRet* =] *object*.**TextAngle**

[**Let**] *object*.**TextAngle** = *textAngleSet*

The **TextAngle** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
textAngleRet	Optional. A **Double** type variable.
textAngleSet	Required. An expression that returns a **Double** value.

Remarks

The **TextAngle** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **TextAngle** as a table parameter, use the **CDPT_TEXTANGLE** constant tag.

The angle values are specified in radians.

See Also	Text property, TextBlock property, TextFlipX property, TextFlipY property, TextGPinX property, TextGPinY property, TextHeight property, TextLPinX property, TextLPinY property TextWidth property

TextBkgnd Property

TextBkgnd Property

Read only. Returns a **Color** object that contains information about the background color of this text block.

Applies to: TextBlock object

Syntax

[[**Set**] *colorRet* =] *object*.**TextBkgnd**

The **TextBkgnd** property syntax has these Elements:

Element	**Description**
object	
colorRet	

Remarks

The **TextBkgnd** property is also a table parameter of the shape which contains the *object* text block, that is, its value can be described by a formula. To work with **TextBkgnd** as a table parameter, use the **CDPT_TEXTBKGND** constant tag.

Example

This example demonstrates how to change background color of a text block in an existing shape.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Change text background color
s.TextBlock.TextBkgnd.Index = 2
' Inform ConceptDraw Engine about changes for re-drawing
s.PropertyChanged(CDPT_TEXTBKGND)
```

See Also Color object, Shape object

TextBlock Property

TextBlock Property

Read-only. Returns a **TextBlock** object that corresponds to the text block of the shape.

Applies to: Shape object, Style object

Syntax
[[**Set**] *textBlockRet* =] *object*.**TextBlock**

The **TextBlock** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
textBlockRet	Optional. A **TextBlock** type variable.

Remarks

Shape object:
A text block describes the region where the shape's text is displayed. Text block settings can be modified even if the shape doesn't contain text.

Style object:
When a style is assigned to a shape, the parameters of the **TextBlock** property of the style are set to the **TextBlock** property of the shape. **TextBlock** is only effective when the **HasTxtblockAttr** property of this style is **True**.

See Also	Character property, HasTxtblockAttr property, Paragraph property, Text property, TextAngle property, TextFlipX property, TextFlipY property, TextGPinX property, TextGPinY property, TextHeight property, TextLPinX property, TextLPinY property TextWidth property, TextBlock object

TextFlipX Property

TextFlipX Property

A **Boolean** type property. Specifies whether or not the shape's text is flipped horizontally. **False** - the text is not flipped. **True** - the text is flipped horizontally.

Applies to: Shape object

Syntax
[[**Let**] *booleanRet* =] *object*.**TextFlipX**

[**Let**] *object*.**TextFlipX** = *textFlipXSet*

The **TextFlipX** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
booleanRet	Optional. A **Boolean** type variable.
textFlipXSet	Required. An expression that returns a **Boolean** value.

Remarks

The **TextFlipX** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **TextFlipX** as a table parameter, use the **CDPT_TEXTFLIPX** constant tag.

See Also Text property, TextAngle property, TextBlock property, TextFlipY property, TextGPinX property, TextGPinY property, TextHeight property, TextLPinX property, TextLPinY property TextWidth property

TextFlipY Property

TextFlipY Property

A **Boolean** type property. Specifies whether or not the shape's text is flipped vertically. **False** - the text is not flipped. **True** - the text is flipped vertically.

Applies to: Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**TextFlipY**

[**Let**] *object*.**TextFlipY** = *textFlipYSet*

The **TextFlipY** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
booleanRet	Optional. A **Boolean** type variable.
textFlipYSet	Required. An expression that returns a **Boolean** value.

Remarks

The **TextFlipY** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **TextFlipY** as a table parameter, use the **CDPT_TEXTFLIPY** constant tag.

See Also	Text property, TextAngle property, TextBlock property, TextFlipX property, TextGPinX property, TextGPinY property, TextHeight property, TextLPinX property, TextLPinY property TextWidth property

TextGPinX Property

TextGPinX Property

A **Double** type property. The X-coordinate of the rotation center of the shape's text box in the coordinate system of the shape.

Applies to: Shape object

Syntax

[[**Let**] *doubleRet* =] *object*.**TextGPinX**

[**Let**] *object*.**TextGPinX** = *textGPinXSet*

The **TextGPinX** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
doubleRet	Optional. A **Double** type variable.
textGPinXSet	Required. An expression that returns a **Double** value.

Remarks

The **TextGPinX** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **TextGPinX** as a table parameter, use the **CDPT_TEXTGPINX** constant tag.

The unit of measure for the coordinates are the internal ConceptDraw units (**InternalUnit**).

See Also	Text property, TextAngle property, TextBlock property, TextFlipX property, TextFlipY property, TextGPinY property, TextHeight property, TextLPinX property, TextLPinY property TextWidth property

TextGPinY Property

TextGPinY Property

A **Double** type property. The Y-coordinate of the rotation center of the shape's text box in the coordinate system of the shape.

Applies to: Shape object

Syntax

[[**Let**] *doubleRet* =] *object*.**TextGPinY**

[**Let**] *object*.**TextGPinY** = *textGPinYSet*

The **TextGPinY** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
doubleRet	Optional. A **Double** type variable.
textGPinYSet	Required. An expression that returns a **Double** value.

Remarks

The **TextGPinY** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **TextGPinY** as a table parameter, use the **CDPT_TEXTGPINY** constant tag.

The unit of measure for the coordinates are the internal ConceptDraw units (**InternalUnit**).

See Also	Text property, TextAngle property, TextBlock property, TextFlipX property, TextFlipY property, TextGPinX property, TextHeight property, TextLPinX property, TextLPinY property TextWidth property

TextHeight Property

TextHeight Property

A **Double** type property. The height of the shape's text box.

Applies to: Shape object

Syntax

[[**Let**] *doubleRet* =] *object*.**TextHeight**

[**Let**] *object*.**TextHeight** = *textHeightSet*

The **TextHeight** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
doubleRet	Optional. A **Double** type variable.
textHeightSet	Required. An expression that returns a **Double** value.

Remarks

The **TextHeight** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **TextHeight** as a table parameter, use the **CDPT_TEXTHEIGHT** constant tag.

The unit of measure for the shape's text box height set by **TextHeight** is the internal ConceptDraw unit (**InternalUnit**).

See Also	Text property, TextAngle property, TextBlock property, TextFlipX property, TextFlipY property, TextGPinX property, TextGPinY property, TextLPinX property, TextLPinY property TextWidth property

TextLPinX Property

TextLPinX Property

A **Double** type property. The X offset of the shape's text box rotation center from the center of the shape's coordinate system.

Applies to: Shape object

Syntax

[[**Let**] *doubleRet* =] *object*.**TextLPinX**

[**Let**] *object*.**TextLPinX** = *textLPinXSet*

The **TextLPinX** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
doubleRet	Optional. A **Double** type variable.
textLPinXSet	Required. An expression that returns a **Double** value.

Remarks

The **TextLPinX** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **TextLPinX** as a table parameter, use the **CDPT_TEXTPINX** constant tag.

Note, that modifying the **TextLPinX** automatically changes the value of the **TextGPinX** property. The unit of measure for the offset are internal ConceptDraw units (**InternalUnit**).

See Also	Text property, TextAngle property, TextBlock property, TextFlipX property, TextFlipY property, TextGPinX property, TextGPinY property, TextHeight property, TextLPinY property TextWidth property

TextLPinY Property

TextLPinY Property

A **Double** type property. The Y offset of the shape's text box rotation center from the center of the shape's coordinate system.

Applies to: Shape object

Syntax

[[**Let**] *doubelRet* =] *object*.**TextLPinY**

object.**TextLPinY** = *textLPinYSet*

The **TextLPinY** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
doubleRet	Optional. A **Double** type variable.
textLPinYSet	Required. An expression that returns a **Double** value.

Remarks

The **TextLPinY** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **TextLPinY** as a table parameter, use the **CDPT_TEXTPINY** constant tag.

Note, that modifying the **TextLPinY** automatically changes the value of the **TextGPinY** property. The unit of measure for the offset are internal ConceptDraw units (**InternalUnit**).

See Also	Text property, TextAngle property, TextBlock property, TextFlipX property, TextFlipY property, TextGPinX property, TextGPinY property, TextHeight property, TextLPinX property, TextWidth property

TextWidth Property

TextWidth Property

A **Double** type property. The width of the shape's text box.

Applies to: Shape object

Syntax

[[**Let**] *doubleRet* =] *object*.**TextWidth**

[**Let**] *object*.**TextWidth** = *textWidthSet*

The **TextWidth** property syntax has these Elements:

Element	**Description**

object	Required. An expression that returns a **Shape** object.
doubleRet	Optional. A **Double** type variable.
textWidthSet	Required. An expression that returns a **Double** value.

Remarks

The **TextWidth** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **TextWidth** as a table parameter, use the **CDPT_TEXTWIDTH** constant tag.

The unit of measure for the shape's text box height set by **TextWidth** is the internal ConceptDraw unit (**InternalUnit**).

See Also Text property, TextAngle property, TextBlock property, TextFlipX property, TextFlipY property, TextGPinX property, TextGPinY property, TextHeight property, TextLPinX property, TextLPinY property

Text Property

Text Property

A **String** type property. A string that contains the text of the shape.

Applies to: Shape object

Syntax

[[**Let**] *textStrRet* =] *object*.**Text**

[**Let**] *object*.**Text** = *textStrSet*

The **Text** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
textStrRet	Optional. A **String** type variable.
textStrSet	Required. An expression that returns a **String** value.

Remarks

The **Text** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **Text** as a table parameter, use the **CDPT_TEXT** constant tag.

See Also	TextAngle property, TextBlock property, TextFlipX property, TextFlipY property, TextGPinX property, TextGPinY property, TextHeight property, TextLPinX property, TextLPinY property TextWidth property

Timeout Property

Timeout Property

A **Long** type property. Time interval in seconds through which there will be the corresponding icon in case of mistake emergence when updating data from a source. By default **Timeout** property is equal 60 seconds.

Applies to: DataSource object

Syntax

[[**Let**] *TimeoutRet* =] *object*.**Timeout**

[**Let**] *object*.**Timeout** = *TimeoutSet*

The **Timeout** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **DataSource** object.
TimeoutRet	Optional. A **Long** type variable.
TimeoutSet	Required. An expression that returns a **Long** value.

Remarks

The **Timeout** property is also a table parameter of the DataSource, that is, its value can be described by a formula. To work with **Timeout** as a table parameter, use the **CDPT_DS_RELIABILITY** constant tag.

Example

```
dim ds as DATASOURCE
ds = thisShape.DATASOURCE(1)
trace ds.Timeout
ds.Timeout = 15
```

```
trace ds.Timeout
or
thisShape.SetPropertyFormula("80", CDPT_DS_RELIABILITY, 1)
trace ds.Timeout
```

See Also DataSource object, Action property, Active property, DataSource property, Refresh property, ShowErrors property, ShowWarnings property

Title Property

Title Property

A **String** value. Gets or sets the title of a document / library.

Applies to: Document object, Library object

Syntax

[[**Let**] *titleRet* =] *object*.**Title**

[**Let**] *object*.**Title** = *titleSet*

The **Title** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object from the **Applies to** list.
titleRet	Optional. A **String** type variable.
titleSet	Required. An expression that returns a **String** value.

Remarks

When *object* is a document, don't confuse the **Title** property with the title, displayed in the window title bar of the document. The name in the window title bar is taken from the filename of the document (Name property). When *object* is a library, the **Title** property corresponds to the name, displayed in the title bar of the library and is not related to the filename of the library. You can change the **Title** property from within ConceptDraw in these dialogs: "File->Document Properties->General" for a document, "Library->Library Properties" - for a library.

Example

The example below contains an document level script. It demonstrates how to view the title of the document by creating a shape with the **Text** property which has the "DocTitle" formula.

```
Dim shp as Shape
' Set the Title property for the document
thisDoc.Title = "OLD document title"
' Draw a shape
Set shp = thisDoc.Page(1).DrawRect(100,100,700,400)
' Set formula for Text property of shape
shp.Text= ""
shp.SetPropertyFormula( "DocTitle", CDPT_TEXT )
shp.RecalcProperty( CDPT_TEXT )
' Change the Title property of the document
MsgBox("Changing the Title property of the document")
thisDoc.Title = "NEW document title"
```

See Also Name property, Author property, Subj property, Company property, Desc property

TopMargin Property

TopMargin Property

A **Single** type property. Specifies the distance between the top border of the text box and the first line of text it contains.

Applies to: TextBlock object

Syntax

[[**Let**] *singleRet* =] *object*.**TopMargin**

[**Let**] *object*.**TopMargin** = *topMarginSet*

The **TopMargin** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **TextBlock** object.
singleRet	Optional. A **Single** type variable.
topMarginSet	Required. An expression that returns a **Single** value.

Remarks

The unit of measure for the **TopMargin** property is **InternalUnit**.

The **TopMargin** property is also a table parameter of the shape, to which the *object* text block belongs - that is, its value can be described by a formula. To work with **TopMargin** as a table parameter, use the **CDPT_TOPMARGIN** constant tag.

Example

This example shows how to increase the distance between the top border of the text box and the first line of text it contains. It assumes the shape exists and contains text.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Increase the distance between the top border of the text box and the text by
20 points.
s.TextBlock.TopMargin = 20
' Inform ConceptDraw Engine about the changes to recalculate and redraw the
document
s.PropertyChanged(CDPT_TOPMARGIN)
```

Top Property

Top Property

Gets or sets a Double value, that represents the coordinate of the top point of a rectangle.

Applies to objects: DRect

Syntax

[[**Let**] *RetVal* =] *object*.**Top**

[**Let**] *object*.**Top** = *SetVal*

Element	**Description**
object	A reference to an instance of the object.
RetVal	A Double type variable.
SetVal	A Double type value.

Example

```
Dim MyObject as new DRect   ' Create an instance of the object
MyObject.Top = 100
```

Top Property (Window object)

Top Property (Window object)

Read-only. A **Long** type property. Returns the Y-coordinate of the top left corner of the window.

Applies to: Window object

Syntax

[[**Let**] *topRet* =] *object*.**Top**

The **Top** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Window** object.
topRet	Optional. A **Long** type variable.

Remarks

Note, that the coordinates of the window position are specified in screen pixels, and the coordinate origin is in the left top corner of the parent window frame. To change the dimensions and position of the window, use the **SetWindowRect** method.

Example

. . . .

See Also Left property, Height property, Width property, SetWindowRect method

Type Property

Type Property (CustomProp object)

An **Byte** value. Returns the type of a custom property.

Applies to: CustomProp object

Syntax

[[**Let**] *type* =] *object*.**Type**

[[**Let**] *object*.**Type =** *type*

The **Type** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **CustomProp** object.
type	Optional. An **Byte** type value.

Remarks

The **Type** property can have the following values:

Value	**Constant**	**Description**
0	cdStringCustomProp	Then the **Value** property represent string data
1	cdNumberCustomProp	Then the **Value** property represent numeric data
2	cdFixedListCustomProp	Then the **Value** property represent string from **Format** property list
3	cdVariableListCustomProp	Then the **Value** property represent string from **Format** property list or any other string
4	cdBooleanCustomProp	Then the **Value** property represent boolean data

Type Property (DataSourceValue Object)

Type Property (DataSourceValue object)

An **Byte** type property. Type of data which are in the section **Value** of the table **Data** of parameters of object.

Applies to: DataSourceValue object

Syntax

[[**Let**] *typeRet* =] *object*.**Type**

[**Let**] *object*.**Type** = *typeSet*

The **Type** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
typeRet	Optional. A **Byte** type variable.
typeSet	Required. An expression that returns a **Byte** value. Sets the new data types from the **Value** table of the object **Data** parameters.

Remarks

The **Type** property can take the following values

Type Property (MenuItem property)

Type Property (MenuItem object)

Read-only. An **Integer** type property. Returns the type of the menu item.

Applies to: MenuItem object

Syntax

[[**Let**] *typeRet* =] *object*.**Type**

The **Type** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **MenuItem** object.
typeRet	Optional. An **Integer** type value.

Remarks

The menu item type is set only once when the menu item is created. The **Type** property can take the following values:

Value	Constant	Description
0	cdMenuItemNormal	A normal menu item, which doesn't contain a submenu and is not a separator.

1	cdMenuItemSeparator	A separator - the line that separates two menu items. For this type of menu item the Caption property doesn't matter because the menu item is displayed as a line.
2	cdMenuItemPopup	A menu item that contains a submenu.

Type Property (Window object)

Type Property (Window object)

Read-only. Returns the window type.

Applies to: Window object

Syntax

[[**Let**] *typeRet* =] *object*.**Type**

The **Type** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Window** object.
typeRetVal	Optional. An **Integer** type variable.

Remarks

The **Type** determines the type of the window contents and respectively defines which properties and methods of the **Window** object are supported in the given instance of the **Window** object. The **Type** property for the **Window** object can take the following values:

Constant	**Value**	**Description**
cdDocView	1	The document editing window (document view).
cdTableView	2	The shape parameter table (table view).
cdLibView	3	The library window (library view).
cdBasicView	4	The ConceptDraw Basic script editor (basic editor view).

See Also State property

UnitIndex Property

UnitIndex Property

A **Long** value. Specifies the units of measure used in the document.

Applies to: Document object

Syntax

[[**Let**] *unitIndexRet* =] *object*.**UnitIndex**

[**Let**] *object*.**UnitIndex** = *unitIndexSet*

The **UnitIndex** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Document** object.
unitIndexRet	Optional. A **Long** type variable.
unitIndexSet	Required. An expression that returns a **Long** value.

Remarks

You can also change the **UnitIndex** property from within ConceptDraw in the "File->Document Properties->Settings" dialog. ConceptDraw Basic has pre-defined constants, which correspond to various units of measure. The **UnitIndex** can take only one of these values. Below is the table which lists the constants:

Constant	**Value**	**Description**
cdYard	0	Yards, decimal.
cdFoot	1	Feet, decimal.
cdFootInch	2	Feet, decimal inches.
cdFootFractInch	3	Feet, fractional inches
cdInch	4	Decimal inches.

cdFractInch	5	Fractional inches.
cdMeter	6	Meters.
cdCm	7	Centimeters.
cdMm	8	Millimeters.
cdKm	9	Kilometers.
cdMile	10	Miles.

Example

This example contains a document-level script. It switches between various units of measure: yards, feet, meters, centimeters, kilometers and miles.

```
thisDoc.UnitIndex = cdYard  ' yards
MsgBox("Units of measure: " & "Yards")
thisDoc.UnitIndex = cdFoot  ' foots
MsgBox("Units of measure: " & "Foots")
thisDoc.UnitIndex = cdInch  ' inches
MsgBox("Units of measure: " & "Inches")
thisDoc.UnitIndex = cdMeter  ' meters
MsgBox("Units of measure: " & "Meters")
thisDoc.UnitIndex = cdCm  ' centimeters
MsgBox("Units of measure: " & "Centimeters")
thisDoc.UnitIndex = cdKm  ' kilometers
MsgBox("Units of measure: " & "Kilometers")
thisDoc.UnitIndex = cdMile  ' miles
MsgBox("Units of measure: " & "Miles")
```

VAlign Property

VAlign Property

A **Byte** type property. Specifies vertical alignment of the text inside this text box.

Applies to: TextBlock object

Syntax

[**Let**] *byteRet* = *object*.**VAlign**

[**Let**] *object*.**VAlign** = *vAlignSet*

The **VAlign** property syntax has these Elements:

Element	Description
object	Required. An expression that returns a **TextBlock** object.
byteRet	Optional. A **Byte** type variable.
vAlignSet	Required. An expression that returns a **Byte** value.

Remarks

The **VAlign** property can take one of these values:

Constant	Value	Description
cdbVertTop	0	Align to the top border.
cdbVertMiddle	1	Align to the center.
cdbVertBottom	2	Align to the bottom border.

The **VAlign** property is also a table parameter of the shape, which contains the *object* text box, that is, its value can be described by a formula. To work with **VAlign** as a table parameter, use the **CDPT_VALIGN** constant tag.

Example

This example demonstrates how to align the shape's text to the top border. It assumes a shape that contains text exists in the document.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Sets alignment to the top border.
s.TextBlock.VAlign = cdbVertTop
' Informs ConceptDraw Engine about the changes for re-drawing
s.PropertyChanged(CDPT_VALIGN)
```

See Also

Shape object

Value Property

Value Property

A String value. Gets or sets the default value.

Applies to objects: CustomProp, DataSourceValue

Syntax

[[**Let**] *RetVal* =] *object*.**Value**

[**Let**] *object*.**Value** = *SetVal*

The **Value** property syntax has these Elements:

Element	Description
object	A reference to an instance of the object.
RetVal	A String type variable.
SetVal	A String value.

Remarks

Below is the meaning of the **Value** property for different objects:

CustomProp	Default Value.
DataSourceValue	Values from the **Value** table of the object **Data** parameters

Example

This example demonstrates working with the **CustomProp** object.

```
Dim MyShape As Shape, MyProperty as  CustomProp
' Create a  Shape
MyShape = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
'  Create custom properties for MyShape
MyProperty = MyShape.AddCustomProp()
'  Working with the properties of MyProperty
MyProperty.Label = "IP"
MyProperty.Prompt = "TCP/IP address"
MyProperty.Type = 3
MyProperty.Format = "192.168.0.1;192.168.0.2;192.168.0.3"
MyProperty.Value = "192.168.0.1"
MyProperty.Invisible = FALSE
MyProperty.Verify = TRUE
```

This example demonstrates working with the **DataSourceValue** object.

```
dim ds as DATASOURCEVALUE
ds = thisShape.DSVALUE(1)
trace ds.value
```

See Also CustomProp Object, DataSourceValue object, Document Object

Verify Property

Verify Property

A Boolean value. Gets or sets the Verify / Not Verify state.

Applies to objects: CustomProp

Syntax

[[**Let**] *RetVal* =] *object*.**Verify**

[**Let**] *object*.**Verify** = *SetVal*

The **Verify** property syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
RetVal	A Boolean type variable.
SetVal	A Boolean value.

Remarks

If **Verify** is TRUE and the Value property is not initialized when a new instance of the Shape property is created, the user will be asked to set the Value property. If **Verify** is FALSE the Value property will not be requested.

Example

This example demonstrates working with the **CustomProp** object.

```
Dim MyShape As Shape, MyProperty as  CustomProp
' Create a  Shape
MyShape = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
'  Create custom properties for MyShape
MyProperty = MyShape.AddCustomProp()
'  Working with the properties of MyProperty
MyProperty.Label = "IP"
MyProperty.Prompt = "TCP/IP address"
MyProperty.Type = 3
MyProperty.Format = "192.168.0.1;192.168.0.2;192.168.0.3"
MyProperty.Value = "192.168.0.1"
MyProperty.Invisible = FALSE
MyProperty.Verify = TRUE
```

See Also CustomProp Object, Document Object

ViewCenterX Property

ViewCenterX Property

Read-only. A **Double** value. Returns the X-coordinate of the point in the center of the window in the coordinate system of the shape or the page, displayed in the window.

Applies to: Window object

Syntax

[[**Let**] *viewCenterXRet* =] *object*.**ViewCenterX**

The **ViewCenterX** property syntax has these Elements:

Element	Description
object	Required. An expression, that returns a **Window** object.
viewCenterXRet	Optional. A **Double** type variable.

Remarks

The unit of measure for the **ViewCenterX** property are the internal ConceptDraw units (**InternalUnit**). To scroll the window to a specified position you can use the **ScrollViewTo** method.

See Also ViewZoom property, ViewCenterY property, ScrollViewTo method

ViewCenterY Property

ViewCenterY Property

Read-only. A **Double** value. Returns the Y-coordinate of the point in the center of the window in the coordinate system of the shape or the page, displayed in the window.

Applies to: Window object

Syntax

[[**Let**] *viewCenterYRet* =] *object*.**ViewCenterY**

The **ViewCenterY** property syntax has these Elements:

Element	Description
object	Required. An expression, that returns a **Window** object.
viewCenterYRet	Optional. A **Double** type variable.

Remarks

The unit of measure for the **ViewCenterY** property are the internal ConceptDraw units (**InternalUnit**). To scroll the window to a specified position you can use the **ScrollViewTo** method.

See Also ViewZoom property, ViewCenterX property, ScrollViewTo method

ViewZoom Property

ViewZoom Property

A **Double** type property. Gets or sets the zoom level for the window.

Applies to: Window object

Syntax

[[**Let**] *viewZoomRet* =] *object*.**ViewZoom**

[**Let**] *object*.**ViewZoom** = *viewZoomSet*

The **ViewZoom** property syntax has these Elements:

Element	**Description**
object	An expression, that returns a **Window** object.
viewZoomRet	A **Long** type variable.
viewZoomSet	An expression that returns a **Long** value.

Remarks

The **ViewZoom** property can have only values greater than **0**. It specifies the ratio of the real and displayed size of the drawing being edited in the active window. When the value equals to 1, the drawing is displayed in its real size. Note, that the view is magnified relative to the center of the window (see the **ViewCenterX**, **ViewCenterY** properties). You can also modify the zoom level using the interface of ConceptDraw.

See Also ViewCenterX property, ViewCenterY property

Visible Property

Visible Property

A **Boolean** type property. Gets or sets the flag that specifies whether the object from the **Applies to** list is visible (**True**) or invisible (**False**).

Applies to: DataSourceValue object, Geometry object, Layer object

Syntax

[[**Let**] *visibleRet* =] *object*.**Visible**

[**Let**] *object*.**Visible** = *visibleSet*

The **Visible** property syntax has these Elements:

Element	**Description**

object	Required. An expression that returns an object in the **Applies to** list.
visibleRet	Optional. A **Boolean** type variable.
visibleSet	Required. An expression that returns a **Boolean** value.

Remarks

Geometry object:
If *object* is a geometry, the **Visible** property determines whether to display this geometry of shape. You can also make the geometry visible or invisible by using the shape parameter table.

The **Visible** property is also a table parameter of the shape which owns the *object* geometry, that is, its value can be described by a formula. To work with **Visible** as a table parameter, use the **CDPT_GEOMETRY_VISIBLE** constant tag.

Layer object:
If *object* is a layer, the value of **Visible** determines whether to display all shapes on that layer of the document. You can also make the layer visible or invisible by using the ConceptDraw dialog "View->Floating Dialogs->Layers".

DataSourceValue object:
Values from the **Value** table of the object **Data** parameters. It is responsible for the displaying Values from the **Value** table of the object **Data** parameters in the corresponding dialog.

See Also Filled property, Shape object

Width Property

Width Property

Read-only. A **Double** type property. The shape's width.

Applies to: Shape object

Syntax

[[**Let**] *widthRet* =] *object*.**Width**

[**Let**] *object*.**Width** = *widthSet*

The **Width** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Window** object.
heightRet	Optional. A **Double** type variable.
heightSet	Required. An expression that returns a **Double** value.

Remarks

The **Width** property is also a table parameter of the shape, that is, its value can be described by a formula. To work with **Width** as a table parameter, use the the **CDPT_WIDTH** constant tag.

The unit of measure for the shape's width set by **Width** is the internal ConceptDraw unit (**InternalUnit**).

See Also	Angle property, GPinX property, GPinY property, FlipX property, FlipY property, Height property, LPinX property, LPinY property, Width property

Width Property (Window object)

Width Property (Window object)

Read-only. A **Long** type property. Returns the width of the window in pixels.

Applies to: Window object

Syntax

[[**Let**] *widthRet* =] *object*.**Width**

The **Width** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Window** object.
widthRet	Optional. A **Long** type variable.

Remarks

Note, that window coordinates and dimensions are measured in screen pixels. To change the position and dimensions of a window, use the **SetWindowRect** method.

See Also Left property, Top property, Height property, SetWindowRect method

XBehaviour Property

XBehaviour Property

This property controls the behavior of the ControlDot object. Controls the type of behavior the x-coordinate of the control handle will exhibit after the handle is moved. Gets or sets a Byte value.

Applies to objects: ControlDot

Syntax

[**Let**] *RetVal* = *object*.**XBehaviour**

[**Let**] *object*.**XBehaviour** = *SetVal*

The **XBehaviour** property syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
RetVal	A Byte type variable.
SetVal	A Byte value.

The possible values of the property are described in the table below:

Constant	**Value**	**Behavior**	**Definition**
cdbCtlProportional	0	Proportional	The control handle can be moved, and it also moves in proportion with the shape when it is stretched.
cdbCtlLocked	1	Proportional locked	The control handle moves in proportion with the shape but the control handle itself cannot be moved.
cdbCtlOffsetMin	2	Offset from left edge	The control handle is offset a constant distance from the left side of the shape.

cdbCtlOffsetMid	3	Offset from center	The control handle is offset a constant distance from the center of the shape.
cdbCtlOffsetMax	4	Offset from right edge	The control handle is offset a constant distance from the right side of the shape.
cdbCtlProportionalHidden	5	Proportional, hidden	Same as 0, but the control handle is not visible.
cdbCtlLockedHidden	6	Proportional locked, hidden	Same as 1, but the control handle is not visible.
cdbCtlOffsetMinHidden	7	Offset from left edge, hidden	Same as 2, but the control handle is not visible.
cdbCtlOffsetMidHidden	8	Offset from center, hidden	Same as 3, but the control handle is not visible.
cdbCtlOffsetMaxHidden	9	Offset from right edge, hidden	Same as 4, but the control handle is not visible.

Example

This example demonstrates using the XBehaviour property.

```
Dim MyControlDot as ControlDot, MyShape As Shape
MyShape = thisDoc.ActivePage.DrawRect(50,50,500,500)    ' Create a Shape
object
MyControlDot = MyShape.AddControlDot()
MyControlDot.X = 100 ' Set ControlDot to specified coordinates
MyControlDot.Y = 150
MyControlDot.XBehaviour = cdbCtlOffsetMin  ' Set  XBehaviour type
' Inform ConceptDraw engine about the changes
MyShape.PropertyChanged(CDPT_CONTROL_X)
MyShape.PropertyChanged(CDPT_CONTROL_Y)
MyShape.PropertyChanged(CDPT_CONTROL_XBEHAVIOUR)
```

See Also ControlDot Object, YBehaviour Property, PropertyChanged Method

XDyn Property

XDyn Property

Gets or sets a Double value that represents the x-coordinate for a control handle's anchor point in local coordinates.

Applies to objects: ControlDot

Syntax

[**Let**] *RetVal* = *object*.**XDyn**

[**Let**] *object*.**XDyn** = *SetVal*

The **XDyn** property syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
RetVal	A Double type variable.
SetVal	A Double value.

Example

This example demonstrates using the **XDyn** property.

```
Dim MyControlDot as ControlDot, MyShape As Shape
' Create a Shape object
MyShape = thisDoc.ActivePage.DrawRect(50,50,500,500)
MyControlDot = MyShape.AddControlDot()
MyControlDot.X = 100 ' Set ControlDot to specified coordinates
MyControlDot.Y = 150
MyControlDot.XDyn = 110
MyControlDot.YDyn = 150
' Inform ConceptDraw engine about the changes
MyShape.PropertyChanged(CDPT_CONTROL_X)
MyShape.PropertyChanged(CDPT_CONTROL_Y)
MyShape.PropertyChanged(CDPT_CONTROL_XDYN)
MyShape.PropertyChanged(CDPT_CONTROL_YDYN)
```

See Also	ControlDot Object, YDyn Property, PropertyChanged Method

X Property

X Property

A **Double** type property. The X-coordinate of the point.

Applies to: ConnectDot object, ControlDot object, DPoint object, Variable object

Syntax

[[**Let**] *xCoordinateRet* =] *object*.**X**

[**Let**] *object*.**X** = *xCoordinateSet*

The **X** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
xCoordinateRet	Optional. A **Double** type variable.
xCoordinateSet	Required. A expression that returns a **Double** value.

See Also LPtoWP method, LPtoGP method, WPtoLP method

YBehaviour Property

YBehaviour Property

This property controls the behavior of the ControlDot object. Controls the type of behavior the x-coordinate of the control handle will exhibit after the handle is moved. Gets or sets a Byte value.

Applies to objects: ControlDot

Syntax

[**Let**] *RetVal* = *object*.**YBehaviour**

[**Let**] *object*.**YBehaviour** = *SetVal*

The **YBehaviour** property syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
RetVal	A Byte type variable.
SetVal	A Byte value.

The possible values of the property are described in the table below:

Constant	**Value**	**Behavior**	**Definition**
cdbCtlProportional	0	Proportional	The control handle can be moved, and it also moves in proportion with the shape when it is stretched.
cdbCtlLocked	1	Proportional locked	The control handle moves in proportion with the shape but the control handle itself cannot be moved.
cdbCtlOffsetMin	2	Offset from bottom edge	The control handle is offset a constant distance from the bottom of the shape.
cdbCtlOffsetMid	3	Offset from center	The control handle is offset a constant distance from the center of the shape.
cdbCtlOffsetMax	4	Offset from top edge	The control handle is offset a constant distance from the top of the shape.
cdbCtlProportionalHidden	5	Proportional, hidden	Same as 0, but the control handle is not visible.
cdbCtlLockedHidden	6	Proportional locked, hidden	Same as 1, but the control handle is not visible.
cdbCtlOffsetMinHidden	7	Offset from bottom edge, hidden	Same as 2, but the control handle is not visible.
cdbCtlOffsetMidHidden	8	Offset from center, hidden	Same as 3, but the control handle is not visible.
cdbCtlOffsetMaxHidden	9	Offset from top edge, hidden	Same as 4, but the control handle is not visible.

Example

This example demonstrates using the YBehaviour property.

```
Dim MyControlDot as ControlDot, MyShape As Shape
MyShape = thisDoc.ActivePage.DrawRect(50,50,500,500)    ' Create a Shape
object
MyControlDot = MyShape.AddControlDot()
MyControlDot.X = 100 ' Set ControlDot to specified coordinates
MyControlDot.Y = 150
MyControlDot.YBehaviour = cdbCtlOffsetMid  ' Set  YBehaviour type
' Inform ConceptDraw engine about the changes
```

```
MyShape.PropertyChanged(CDPT_CONTROL_X)
MyShape.PropertyChanged(CDPT_CONTROL_Y)
MyShape.PropertyChanged(CDPT_CONTROL_YBEHAVIOUR)
```

See Also ControlDot Object, XBehaviour Property, PropertyChanged Method

YDyn Property

YDyn Property

Gets or sets a Double value that represents the Y-coordinate for a control handle's anchor point in local coordinates.

Applies to objects: ControlDot

Syntax

[**Let**] *RetVal* = *object*.**YDyn**

[**Let**] *object*.**YDyn** = *SetVal*

The **YDyn** property syntax has these Elements:

Element	Description
object	A reference to an instance of the object.
RetVal	A Double type variable.
SetVal	A Double value.

Example

This example demonstrates using the **YDyn** property.

```
Dim MyControlDot as ControlDot, MyShape As Shape
'  Create a Shape object
MyShape = thisDoc.ActivePage.DrawRect(50,50,500,500)
MyControlDot = MyShape.AddControlDot()
MyControlDot.X = 100 '  Set ControlDot to specified coordinates
MyControlDot.Y = 150
MyControlDot.XDyn = 110
MyControlDot.YDyn = 150
' Inform ConceptDraw engine about the changes
MyShape.PropertyChanged(CDPT_CONTROL_X)
MyShape.PropertyChanged(CDPT_CONTROL_Y)
MyShape.PropertyChanged(CDPT_CONTROL_XDYN)
MyShape.PropertyChanged(CDPT_CONTROL_YDYN)
```

See Also ControlDot Object, XDyn Property, PropertyChanged Method

Yellow Property

Yellow Property

Gets or sets an **Integer** value, that represents the yellow component of CMYK color.

Applies to: Color object, ColorEntry object

Syntax

[[**Let**] *yellowRet* =] *object*.**Yellow**

[**Let**] *object*.**Yellow** = *yellowSet*

The **Yellow** property syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object from the **Applies to** list.
yellowRet	Optional. An **Integer** type variable.
yellowSet	Required. An expression that returns an **Integer** value.

Remarks

The **Yellow** property is only effective if the color is a CMYK color (see the **IsCMYK** property).

Example

This example contains a document-level script. It demonstrates how to find out the value of the yellow component of the fill color (in CMYK format) of a Shape object.

```
dim s as shape
' ShapeObject creation
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
If s.FillColor.IsCMYK<> false  Then    ' A CMYK color?
 MsgBox(s.FillColor.Yellow)    ' If yes, display the value of the cyan
component.
endif
```

See Also Cyan property, Magenta property, Black property, IsCMYK property

Y Property

Y Property

A **Double** type property. The Y-coordinate of the point.

Applies to: ConnectDot object, ControlDot object, DPoint object, Variable object

Syntax

[[**Let**] *yCoordinateRet* =] *object*.**Y**

[**Let**] *object*.**Y** = *yCoordinateSet*

The **X** property syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
xCoordinateRet	Optional. A **Double** type variable.
xCoordinateSet	Required. A expression that returns a **Double** value.

See Also LPtoWP method, LPtoGP method, WPtoLP method

ConceptDraw access Objects methods

ActionsNum Method

ActionsNum Method

Returns the number of user-defined actions of the shape.

Applies to: Shape object

Syntax

[[**Let**] *countRet* =] *object*.**ActionsNum** ()

The **ActionsNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
countRet	Optional. A **Long** type variable.

Remarks

If the shape doesn't contain any user-defined actions, the **ActionsNum** method returns **0**.

See Also	Action method, ActionsNum method, AddAction method, RemoveAction method

Action Method

Action Method

Returns an instance of the **Action** object that corresponds to the user-defined action and the associated menu item by its index in the user-defined action collection of the shape.

Applies to: Shape object

Syntax

[[**Set**] *actionRet* =] *object*.**Action** (*index*)

The **Action** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the user-defined action in the user-defined action collection of the shape.
actionRet	Optional. An **Action** type variable.

Remarks

If *index* is less than **1** or greater than the number of user-defined actions of the shape, the **Action** method returns **Nothing**. To find out the number of user-defined actions of the shape, use the **ActionsNum** method.

See Also Action method, ActionsNum method, AddAction method, RemoveAction method, Action object

AddAction Method

AddAction Method

Adds a new user-defined action to the user-defined action collection of the shape.

Applies to: Shape object

Syntax

[[**Let**] *actionRet* =] *object*.**AddAction** ()

The **AddAction** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
actionRet	Optional. An **Action** type variable.

Remarks

If the action was added successfully, the **AddAction** method returns the **Action** object that corresponds to the added user-defined action. Otherwise the method returns **Nothing**.

See Also Action method, ActionsNum method, AddAction method, RemoveAction method, Action object

AddConnectDot Method

AddConnectDot Method

Adds a new connection point to the connection point collection of the shape. Returns the **ConnectDot** object that corresponds to the added connection point.

Applies to: Shape object

Syntax

[[**Set**] *connectDotRet* =] *object*.**AddConnectDot** ()

The **AddConnectDot** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
connectDotRet	Optional. A **ConnectDot** type variable.

Remarks

Note that by default a new connection point is located in the point with (0,0) coordinates in the coordinate system of the *object* shape, to which this connection point is added. To change the position of the connection point use the **X** and **Y** properties of the **ConnectDot** object.

See Also	ConnectDot method, ConnectDotsNum method, RemoveConnectDot method, ConnectDot object

AddControlDot Method

AddControlDot Method

Adds a new control handle to the control handle collection of the shape. Returns a **ControlDot** object that corresponds to the added control handle.

Applies to: Shape object

Syntax

[[**Set**] *controlDotRet* =] *object*.**AddControlDot** ()

The **AddControlDot** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
controlDotRet	Optional. A **ControlDot** type variable.

Remarks

Note that by default a new control handle is located in the point with (0,0) coordinates in the coordinate system of the *object* shape, to which this control handle is added. To change the position of the control handle use the **X** and **Y** properties of the **ControlDot** object.

See Also ControlDot method, ControlDotsNum method, RemoveControlDot method, ControlDot object

AddCustomProp Method

AddCustomProp Method

Adds a new custom property of the shape to the custom property collection of the shape. Returns a **CustomProp** object that corresponds to the added custom property.

Applies to: Shape object

Syntax

[[**Set**] *customPropRet* =] *object*.**AddCustomProp** ()

The **AddCustomProp** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
customPropRet	Optional. A **CustomProp** type variable.

Remarks

If the custom property couldn't be added, the **AddCustomProp** method returns **Nothing**.

See Also	AddCustomProp method, CustomProp method, CustomPropByLabel, CustomPropsNum method, RemoveCustomProp method, CustomProp object

AddDataSource Method

AddDataSource Method

Adds a new data source to the collection of data source object (shape). Returns an instance of DataSource, corresponding to the new you added the source.

Applies to: Shape object

Syntax

[[**Set**] *dataSourceRet* =] *object*.**AddDataSource**()

The **AddDataSource** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dataSourceRet	Optional. A **DataSource** type variable.

Remarks

AddDataSource method in case of failure returns.

Example

```
dim num as Integer
dim ds as DataSource
num = thisShape.DataSourcesNum()
trace num
ds = thisShape.AddDataSource()
trace ds.Refresh
num = thisShape.DataSourcesNum()
trace num
```

See Also	DataSource object, DataSource method, DataSourcesNum method, RemoveDataSource method

AddDSValue Method

AddDSValue Method

Adds a new row containing the field Value, in the Data Table parameters of the object (shape). Returns an instance of an object DataSourceValue, containing data in the inserted rows.

Applies to: Shape object

Syntax

[[**Set**] *dataSourceValueRet* =] *object*.**AddDSValue**()

The **AddDSValue** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dataSourceValueRet	Optional. A **DataSourceValue** type variable.

Remarks

AddDataSource method in case of failure returns 0.

Example

```
dim num as Integer
dim ds as DataSourceValue
num = thisShape.DSValuesNum()
trace num
ds = thisShape.AddDSValue()
num = thisShape.DSValuesNum()
trace num
```

See Also	DataSourceValue object, DSValue method, DSValueEl method, DSValuesNum method, RemoveDSValue method

AddGeometry Method

AddGeometry Method

Adds a new geometry to the geometry collection of the shape. Returns a **Geometry** object that corresponds to the added geometry.

Applies to: Shape object

Syntax

[[**Set**] *geometryRet* =] *object*.**AddGeometry** (*xStart*, *yStart*)

The **AddGeometry** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
xStart	Required. An expression that returns a **Double** value. The X-coordinate of the start segment of the geometry.
yStart	Required. An expression that returns a **Double** value. The Y-coordinate of the start segment of the geometry.
customPropRet	Optional. A **CustomProp** type variable.

Remarks

The **AddGeometry** method adds a new geometry that contains the start segment in the point with the *xStart* and *yStart* coordinates in the coordinate system of *object*. In case the new geometry couldn't be added, the **AddGeometry** method returns **Nothing**.

See Also GeometriesNum method, Geometry method, RemoveGeometry method, Geometry object

AddHyperlinkToDocument Method

AddHyperlinkToDocument Method

Adds a hyperlink pointing to a ConceptDraw to the hyperlink collection of the document. Returns the ID (the **ID** property) of the added hyperlink.

Applies to: Document object

Syntax

[[**Let**] *linkIDRet* =] *object*.**AddHyperlinkToDocument** (*fileName*, [*localPath*], [*pageID*], [*shapeID*])

The **AddHyperlinkToDocument** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Document** object.
fileName	Required. An expression that returns a **String** value. Specifies the filename (with the full or relative path) to which the added hyperlink will point.
localPath	Optional. An expression that returns a **Boolean** value. If *localPath* is **True**, then the *fileName* represents a relative path (with respect to the folder, in which the document is located). Otherwise *fileName* contains the full path to the file. The default value is **False**.
pageID	Optional. An expression that returns a **Long** value. Represents the ID of the document page to which the added hyperlink will point. The default value is **0**, which means the hyperlink doesn't point to any specific page.
shapeID	Optional. An expression that returns a **Long** value. Represents the ID of the shape to which the added hyperlink will point. The default value is **0**, which means the hyperlink doesn't point to any specific shape.
linkIDRet	Optional. A **Long** type variable.

Remarks

If the hyperlink was added successfully, the **AddHyperlinkToDocument** method returns the ID of the added hyperlink. If the hyperlink collection of the document already contains a hyperlink with the same properties, the method doesn't create a new hyperlink, but returns the ID of the identical hyperlink. In all other cases the method returns **0**.

Note, that the method can't add a hyperlink with no filename. That is, *fileName* must contain at least one character.

A hyperlink created with the **AddHyperlinkToDocument** method has the **cdLinkToFile** type (see the LinkType property).

Example

This example contains a document-level script. The program creates a rectangle that contains a hyperlink pointing to a ConceptDraw document, chosen by the user. The hyperlink is added by using the **AddHyperlinkToDocument** method. In order to see the result of this example, the user needs to point the hyperlink to a ConceptDraw document file, and specify the page and shape IDs to which the hyperlink will point.

```
' Declare variables
Dim shp As Shape
Dim linkID As Long
Dim pageID As Long
Dim shapeID As Long
Dim fileName As String
' Get the attributes needed to create hyperlink:
    ' Get file name
    fileName = GetOpenFileName( "cdd", ,"Choose file!" )
if fileName <> "" AND fileName <> Null Then
    ' Get page ID
    pageID = InputBox( "Enter page ID:" )
    ' Get shape ID
    shapeID= InputBox( "Enter shape ID:" )
    ' Add hyperlink using the provided filename
    Let linkID = thisDoc.AddHyperlinkToDocument( fileName, True, pageID,
shapeID )
    ' Draw rectangle
    Set shp = thisDoc.ActivePage.DrawRect( 100,100,700,500 )
    ' Assign text to rectangle
    shp.Text = fileName
    ' Assign hyperlink to rectangle
    shp.Hyperlink = linkID
    ' Set double-click action to open hyperlink
    shp.DblClick = 4
Else
    MsgBox( "You did not choose any file!" )
End If
```

See Also	ID property, LinkType property, AddHyperlinkToFile method, AddHyperlinkToPageShape method, AddHyperlinkToURL method, Hyperlink method, HyperlinkByID method, HyperlinksNum method, RemoveUnusedHyperlinks method, Hyperlink object

AddHyperlinkToFile Method

AddHyperlinkToFile Method

Adds a hyperlink pointing to a file to the hyperlink collection of the document. Returns the ID (**ID** property) of the added hyperlink.

Applies to: Document object

Syntax

[[**Let**] *linkIDRet* =] *object*.**AddHyperlinkToFile** (*filename*, [*localPath*])

The **AddHyperlinkToFile** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Document** object.
fileName	Required. An expression that returns a **String** value. Specifies the filename (with the full or relative path) to which the added hyperlink will point.
localPath	Optional. An expression that returns a **Boolean** value. If *localPath* is **True**, then the *fileName* represents a relative path (with respect to the folder, in which the document is located). Otherwise *fileName* contains the full path to the file. The default value is **False**.
linkIDRet	Optional. A **Long** type variable.

Remarks

If the hyperlink was added successfully, the **AddHyperlinkToFile** method returns the ID of the added hyperlink. If the hyperlink collection of the document already contains a hyperlink with the same properties, the method doesn't create a new hyperlink, but returns the ID of the identical hyperlink. In all other cases the method returns **0**.

A hyperlink created with the **AddHyperlinkToFile** method has the **cdLinkToFile** type (see the LinkType property).

Example

This example contains a document-level script. The program creates a rectangle that contains a hyperlink pointing to a file, chosen by the user. The hyperlink is added by using the **AddHyperlinkToFile** method.

```
' Declare variables
Dim shp As Shape
Dim linkID As Long
Dim fileName As String
' Get the name of the file
fileName = GetOpenFileName( ,,"Choose file!" )
if fileName <> "" AND fileName <> Null Then
    ' Add hyperlink using the provided filename
    Let linkID = thisDoc.AddHyperlinkToFile( fileName )
    ' Draw rectangle
    Set shp = thisDoc.ActivePage.DrawRect( 100,100,700,500 )
    ' Assign text to rectangle
    shp.Text = fileName
    ' Assign hyperlink to rectangle
    shp.Hyperlink = linkID
    ' Set double-click action to open hyperlink
    shp.DblClick = 4
```

```
Else
    MsgBox( "You did not choose any file!" )
End If
```

See Also ID property, LinkType property, AddHyperlinkToDocument method, AddHyperlinkToPageShape method, AddHyperlinkToURL method, Hyperlink method, HyperlinkByID method, HyperlinksNum method, RemoveUnusedHyperlinks method, Hyperlink object

AddHyperlinkToPageShape Method

AddHyperlinkToPageShape Method

Adds a hyperlink pointing to a page or a shape located on the specified page of the ConceptDraw document to the hyperlink collection of the document. Returns the ID (**ID** property) of the added hyperlink.

Applies to: Document object

Syntax

[[**Let**] *linkIDRet* =] *object*.**AddHyperlinkToPageShape** (*pageID*, [*shapeID*])

The **AddHyperlinkToPageShape** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Document** object.
pageID	Required. An expression that returns a **Long** value. Specifies the ID (the ID property) of the page, to which the added hyperlink will point.
shapeID	Optional. An expression that returns a **Long** value. Represents the ID of the shape to which the added hyperlink will point. The default value is **0**, which means the hyperlink doesn't point to any specific shape.
linkIDRet	Optional. A **Long** type variable.

Remarks

If the hyperlink was added successfully, the **AddHyperlinkToPageShape** method returns the ID of the added hyperlink. If the hyperlink collection of the document already contains a hyperlink with the same properties, the method doesn't create a new hyperlink, but returns the ID of the identical hyperlink. In all other cases the method returns **0**.

A hyperlink created with the **AddHyperlinkToPageShape** method has the **cdLinkToPageShape** type (see the LinkType property).

Example

This example contains a document-level script. The program creates the header and the footnote on the active page of the document, represented by two rectangles at the top and bottom of the page. Each rectangle has a hyperlink, pointing to the other rectangle. The hyperlinks are added by using the **AddHyperlinkToPageShape** method.

```
' Declare variables
Dim a_page As Page
Dim header As Shape
Dim footer As Shape
Dim linkid1 As Integer
Dim linkid2 As Integer
' Get the active page
Set a_page = thisDoc.ActivePage
' Create the header for the page
Set header = a_page.DrawRect( 0, -50, thisDoc.PageSizeX, 0)
' Create the footnote for the page
Set footer = a_page.DrawRect( 0, thisDoc.PageSizeY, thisDoc.PageSizeX,
thisDoc.PageSizeY+50 )
' Add hyperlink pointing to header
Let linkid1 = thisDoc.AddHyperlinkToPageShape( a_page.ID, header.ID )
' Add hyperlink pointing to header
Let linkid2 = thisDoc.AddHyperlinkToPageShape( a_page.ID, footer.ID )
' Assign hyperlinks to shapes
Let header.Hyperlink = linkid2
Let footer.Hyperlink = linkid1
header.DblClick = 4
footer.DblClick = 4
header.Text = "PAGE START - Double click to go to the end of page"
footer.Text = "PAGE END - Double click to go to the beginning of page"
```

See Also	ID property, LinkType property, AddHyperlinkToDocument method, AddHyperlinkToFile method, AddHyperlinkToURL method, Hyperlink method, HyperlinkByID method, HyperlinksNum method, RemoveUnusedHyperlinks method, Hyperlink object

AddHyperlinkToURL Method

AddHyperlinkToURL Method

Adds a hyperlink pointing to URL (an Internet address) to the hyperlink collection of the document. Returns the ID (the **ID** property) of the added hyperlink.

Applies to: Document object

Syntax

[[**Let**] *linkIDRet* =] *object*.**AddHyperlinkToURL** (*url*)

The **AddHyperlinkToURL** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the **Document** object.
url	Required. An expression that returns a **String** value. Specifies the URL address of the resource to which the added hyperlink will point.
linkIDRet	Optional. A **Long** type variable.

Remarks

If the hyperlink was added successfully, the **AddHyperlinkToURL** method returns the ID of the added hyperlink. If the hyperlink collection of the document already contains a hyperlink with the same properties, the method doesn't create a new hyperlink, but returns the ID of the identical hyperlink. In all other cases the method returns **0**.

A hyperlink created with the **AddHyperlinkToURL** method has the **cdLinkToURL** type (see the LinkType property).

Example

This example contains a document-level script. The program adds a hyperlink pointing to the ConceptDraw web site (www.conceptdraw.com) to the hyperlink collection of the document. Then the user can input the ID of the shape in the document, to which the created hyperlink will be assigned. The hyperlink is added by using the **AddHyperlinkToURL** method.**AddHyperlinkToURL**.

```
' Declare variables
Dim shp As Shape
Dim cur_page As Page
Dim shapeID As Long
Dim hlinkID As Long
' Add hyperlink pointing to the ConceptDraw web site to the
' hyperlink collection of the document
hlinkID = thisDoc.AddHyperlinkToURL( "www.conceptdraw.com" )
' Ask the user to input the ID of the shape to which the hyperlink will be
assigned.
shapeID = InputBox( "Enter the ID of the shape to assign the hyperlink to:" )
' Loop through all pages of the document, until the shape with the provided ID
is found.
' Assign the hyperlink to that shape.
For i=1 To thisDoc.PagesNum()
    ' Look for the shape with provided ID in the shape collection of the
document.
    Set shp = thisDoc.Page(i).ShapeByID( shapeID )
    ' If the shape is found, assign the hyperlink to it.
    If shp <> Null Then
        shp.Hyperlink = hlinkID
        shp.DblClick = 4
    End If
Next i
```

See Also	ID property, LinkType property, AddHyperlinkToDocument method, AddHyperlinkToFile method, AddHyperlinkToPageShape method, Hyperlink method, HyperlinkByID method, HyperlinksNum method, RemoveUnusedHyperlinks method, Hyperlink object

AddLayer Method

AddLayer Method

Adds a new layer to the layer collection of the document. Returns a **Layer** object corresponding to the created layer.

Applies to: Document object

Syntax

[[**Set**] *layerRet* =] *object*.**AddLayer** ()

The **AddLayer** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns a **Document** object.
layerRet	Optional. A **Layer** type variable.

Remarks

The layer created by using the **AddLayer** method is added at the end of the layer collection of the document. All properties of the new layer get the default values. To change the properties of the layer, use the properties and methods of the **Layer** object.

Example

This example contains a document-level script. It uses the **AddLayer** method to add a new layer. Then this layer is made active and a complex shape is drawn on it.

```
' Declare variables
Dim new_layer As Layer
```

```
' Add new layer to document
Set new_layer = thisDoc.AddLayer()
' Display the name of the new layer
TRACE new_layer.Name
' Make the layer colored.
new_layer.Colored = TRUE
' Set the current layer color to blue.
new_layer.Color.SetRGB(0,0,255)
' Make the new layer active
thisDoc.ActiveLayer = new_layer.ID
'''''''''''''''''''''''''''''''''''''''''''''
' Draw some figure on the new layer
' The figure takes the color of the layer on which it's being drawn
dy = 10
smax = 700 / dy
x1 = 0
y1 = 0
x2 = 0
y2 = 0
For i=1 To smax
    thisDoc.ActivePage.DrawLine( x1,y1,x2,y2 )
    y1 = y1 + dy
    x1 = sqr( y1*200 )
    x2 = sqr( y1*600 )
    y2 = y1
Next i
x1 = 900
y1 = 0
x2 = 900
y2 = 0
For i=1 To smax
    thisDoc.ActivePage.DrawLine( x1,y1,x2,y2 )
    y1 = y1 + dy
    x1 = 1000 - sqr( y1*200 )
    x2 = 1000 - sqr( y1*600 )
    y2 = y1
Next i
''''''''''''''''''''''''''''''''''''''''''''
```

See Also	Layer method, LayerByID method, LayerByName method, LayersNum method, RemoveLayer method, RemoveLayerByID method, Layer object

AddMaster Method

AddMaster Method

Adds to the library a new master object, based on the specified existing shape. Returns a **Master** object that corresponds to the added master object.

Applies to: Library object

Syntax

[[**Set**] *masterRet* =] *object*.**AddMaster** (*shapeSrc*)

The **AddMaster** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Library** object.
shapeSrc	Required. An expression that returns a **Shape** object. The shape on which the new master object is based.
masterRet	Optional. A **Master** type variable.

Remarks

If the specified *shapeSrc* shape couldn't be copied, the **AddMaster** method doesn't add the new master object and returns **Nothing**.

See Also	AddMaster method, FindMaster method, Master method, MasterByName method, MastersNum method, RemoveMaster method, RemoveMasterByName method

AddMenuItem Method

AddMenuItem Method

Adds a new menu item to the menu.

Applies to: Menu object

Syntax

[[**Set**] *menuItemRet* =] *object*.**AddMenuItem** (*menuItemType*)

The **AddMenuItem** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Menu** object.

menuItemType	Required. An expression that returns an **Integer** value. Specifies the type of the menu item (the **Type** property).
menuItemRet	Optional. A **MenuItem** type variable.

Remarks

The *menuItemType* parameter specifies which type (the **Type** property) will the added menu item have.

See Also Type property, MenuItem object

AddPage Method

AddPage Method

Adds a new page to the page collection of the document. Returns an instance of the **Page** object, corresponding to the added page.

Applies to: Document object

Syntax

[[**Set**] *pageRet* =] *object*.**AddPage** ()

The **AddPage** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Document** object.
pageRet	Optional. A **Page** type variable.

Remarks

If for some reason the page couldn't be added to the document, the **AddPage** method returns **Nothing**. Note that a new page, created with **AddPage**, gets default properties. To change the properties of the page use the methods and properties of the **Page** object.

Example

The example below contains a document-level script. It demonstrates how the **AddPage** method is used to add a page to the document. Then an ellipse is created on the page, containing the name of the new page.

```
' Declare variables
Dim new_page As Page
Dim shp As Shape
' Add a new page to the document
Set new_page = thisDoc.AddPage()
' Make the page active
thisDoc.SetActivePageByID( new_page.ID )
' Draw an ellipse on the new page
Set shp = new_page.DrawOval( 100,100,1000,500 )
' Assign the page name as the text of the ellipse
shp.Text = ""
shp.SetPropertyFormula( "_PAGENAME()", CDPT_TEXT )
shp.RecalcProperty( CDPT_TEXT )
```

See Also	FindPage method, Page method, PageByID method, PagesNum method, RemovePage method, RemovePageByID method, ReorderPage method, ReorderPageByID method, Page object

AddStyle Method

AddStyle Method

Adds a new style to the style collection of the document. Returns a **Style** object that corresponds to the new created style.

Applies to: Document object

Syntax

[[**Set**] *styleRet* =] *object*.**AddStyle** (*styleName*)

The **AddStyle** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Document** object.
styleName	Optional. An expression that returns a **String** value. The name (the **Name** property) of the new style.
styleRet	Optional. A **Style** type variable.

Remarks

The **AddStyle** method creates a new style based on the default style (the **DefStyle** property). If there already is a style with the *styleName* name in the collection, the method doesn't create a new style and returns **Nothing**. If *styleName* is not specified, the **AddStyle** method assigns a unique name (the **Name** property) to the style automatically. The new style, created with the **AddStyle** method, is added to the end of the style collection of the document.

Example

This example contains a document-level script. It demonstrates using the **AddStyle** method. It also shows, that it's not possible to add to the collection two styles with the same names.

```
' Declare variables
Dim new_style As Style
' Add a style with some name
Set new_style = thisDoc.AddStyle()
' Display the name of the new style
TRACE new_style.Name
' Add a new style with a specified name
Set new_style = thisDoc.AddStyle("Style_Name_1")
' Display the name of the reference to the instance of the
' object corresponding to the new style
TRACE new_style
' Attempt to add a style with the same name
Set new_style = thisDoc.AddStyle("Style_Name_1")
' Display the name of the reference to the instance of the
' object corresponding to the new style, to make sure the style
' hasn't been added
TRACE new_style
```

See Also	Name property, FindStyle method, RemoveStyle method, RemoveStyleByName method, RenameStyle method, Style method, StyleByName method, StylesNum method, Style object

AddTabStop Method

AddTabStop Method

Adds a tab stop and returns a reference to it.

Applies to: TextBlock object

Syntax

[[**Set**] *ret* =] *object*.**AddTabStop** ()

The **AddTabStop** method syntax has these Elements:

Element	Description
object	A reference to an instance of the object.
ret	A TabStop type variable

Example

This example demonstrates how a tab stop can be added to a shape. It assumes the active page contains a shape with the ID 1, and the shape contains text.

```
Dim s as Shape, MyTabStop as TabStop
s = thisDoc.ActivePage.ShapeByID(1)
Set MyTabStop = s.TextBlock.TabStop(1)
```

See Also	RemoveTabStop method, TabStop method, TabStopsNum method, TabStop object

AddVariable Method

AddVariable Method

Adds a new user-defined variable to the variable collection of the shape. Returns a **Variable** object that corresponds to the added variable.

Applies to: Shape object

Syntax

[[**Set**] *variableRet* =] *object*.**AddVariable** ()

The **AddVariable** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape**.
variableRet	Optional. A **Variable** type variable.

Remarks

If the new variable couldn't be added to the collection, the method returns **Nothing**.

See Also	Variable method, VariablesNum method, RemoveVariable method, Variable object

ArcTo Method

ArcTo Method

Builds an arc of a circle. Returns an instance of the **Shape** object, corresponding to the built shape.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**ArcTo** (*xEnd*, *yEnd*, xMiddle, *yMiddle*)

The **ArcTo** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
xMiddle	Required. An expression that returns a **Double** value. Represents the X-coordinate of the point, that lies on the arc being built.
yMiddle	Required. An expression that returns a **Double** value. Represents the Y-coordinate of the point, that lies on the arc being built.
xEnd	Required. An expression that returns a **Double** value. Represents the X-coordinate of the end of the arc being built.
yEnd	Required. An expression that returns a **Double** value. Represents the Y-coordinate of the end of the arc being built.
shapeRet	Optional. A **Shape** type variable.

Remarks

The arc of a circle is based on three points: the beginning of the arc, a point on the arc, and the end of the arc.

If *object* is a page or a group, the **ArcTo** method creates the arc in the current Basic-shape (the **BeginShape** method), and returns an instance of the **Shape** object, corresponding to that shape. If the method was called prior to the **BeginShape** method or after the **EndShape** method, the **ArcTo** method doesn't create anything and returns **Nothing**.

If *object* is a regular shape, the **ArcTo** method for the shape adds a new arc geometry to the shape and returns *object*.

In any case, the begin point of the arc is the end point of the last geometry of the shape, in which the segment is being built. To reposition the begin point of the arc, use the **MoveTo** method. The coordinates of the points are in the coordinate system of the shape, group or the page - depending on the *object* type. The unit of measure of the specified coordinates is **InternalUnit**.

See Also	BeginShape method, EndShape method, LineTo method, MoveTo method, SplineStart method, SplineTo method

BeginShape Method

BeginShape Method

Returns an instance of the **Shape** object which corresponds to the current Basic shape for the specified page or group.

Applies to: Page object, Shape object

Syntax

[[**Set**] *currentBasicShapeRet* =] *object*.**BeginShape** ()

The **BeginShape** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
currentBasicShapeRet	Optional. A **Shape** type variable.

Remarks

This method is only effective for pages and shapes that are groups. For all other shapes this method returns **Nothing**.

If the **BeginShape** method is called at the first time or after the **EndShape** method has been called, the method creates on the page or in the group a shape that is considered as the current Basic shape for the page or group. On each subsequent call **BeginShape** returns an already existing current Basic shape.

See Also EndShape method

CharactersNum Method

CharactersNum Method

Returns the number of character blocks associated with the shape's text.

Applies to: Shape object

Syntax

[[**Let**] *countRet* =] *object*.**CharactersNum** ()

The **CharactersNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Shape** object.
countRet	Optional. A **Character** type variable.

Remarks

If the shape doesn't contain text, it doesn't contain any character block, so in this case the **CharactersNum** method returns **0**.

See Also Character method, GetCharacterIndex method, RemoveCharacter method, SetCharColor method, SetCharFont method, SetCharHyperlink method, SetCharLanguage method, SetCharPos method, SetCharSize method, SetCharSpacing method, SetCharStyle method, Character object

Character Method

Character Method

Returns a **Character** object, that corresponds to a character block by its index in the character block collection of the shape.

Applies to: Shape object

Syntax

[[**Set**] *characterRet* =] *object*.**Character** (*index*)

The **Character** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the character block in the character block collection of the shape.
characterRet	Optional. A **Character** type variable.

Remarks

If *index* is less than **1** or greater than the number of character blocks of the shape, the **Character** method returns **Nothing**. To find out the number of character blocks in the shape, use the **CharactersNum** method.

See Also CharactersNum method, GetCharacterIndex method, RemoveCharacter method, SetCharColor method, SetCharFont method, SetCharHyperlink method, SetCharLanguage method, SetCharPos method, SetCharSize method, SetCharSpacing method, SetCharStyle method, Character object

CloseDoc Method

CloseDoc Method

Closes the document and all its windows.

Applies to: Application object

Syntax

[[**Let**] *booleanRet* =] *object*.**CloseDoc** (*documentObj*)

The **CloseDoc** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Application** object.
documentObj	Required. An expression that returns an instance of the **Document** object (the document to be closed).
booleanRet	Optional. A **Boolean** type variable.

Remarks

If the document was closed successfully the **CloseDoc** method returns **True**, otherwise it returns **False**. The document can't be closed if a Basic script of the document or one of its shapes or pages is running.

When using the **ClodeDoc** method remember that it closes the *closeDocument* document without saving unsaved changes.

Example

This example contains an application-level script. The script closes all the documents open in the application without saving changes.

```
Dim curDoc as Document             ' Declare the curDoc variable.
 For i=1 To thisapp.DocsNum()       ' Loop through all open documents
    curDoc = thisApp.Doc(1)        ' Get the document with index  1
    thisapp.CloseDoc( curDoc )     'Close the document
Next i
```

See Also	CreateNewDoc method, Doc method, DocByName method, DocsNum method, FirstDoc method, NextDoc method, OpenDoc method, Document object

CloseLib Method

CloseLib Method

Closes a previously opened library.

Applies to: Application object

Syntax

[[**Let**] *booleanRet* =] *object*.**CloseLib**(*libraryObj*)

The **CloseDoc** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Application** object.
libraryObj	Required. An expression that returns an instance of the **Library** object (the library to be closed).
booleanRet	Optional. A **Boolean** type variable.

Remarks

Note, that *libraryObj* must specify an open library, otherwise callingg this method may cause run-time errors. If the library window contains only one open library, closing the library also closes the library window. If the library was closed successfully, the method returns **True**. Otherwise (for instance, if the library has been already closed) it returns **False**.

Example

This example contains an application-level script. It demonstrates using the **CloseLib** method. The script closes all the libraries, open in the application, except for the active library. If there is no active library, none of the libraries is closed.

```
Dim active_lib as Library, current_lib as Library
Dim lib_count as Integer
Set active_lib = thisApp.ActiveLib           ' Get active library
If active_lib <> Nothing Then                ' if we have active library
    lib_count = thisApp.LibsNum()
    For i=lib_count To 1 Step -1             ' loop by every library
        Set current_lib = thisApp.Lib(i)
        If current_lib <> active_lib Then    ' if library is not active
            thisApp.CloseLib( current_lib )  ' then close library
        End If
    Next i
    End                                      ' End script
End If
MsgBox( "There is no active library now." )
```

See Also CloseLib method, CreateNewLib method, FindLib method, Lib method, LibByName method, LibsNum method, OpenLib method, Library object

ColCount Method

ColCount Method

Returns the maximum number of columns in the search for all rows in a table view CSV file data source.

Applies to: DataSource object

Syntax

[[Let] *countRet* =] *object.* **ColCount** ()

The **ColCount** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **DataSource** object.
countRet	Optional. A **Long** type variable.

Remarks

An instance of the **DataSource** object can be obtained using methods of the **Shape.**

Example

```
dim ds as DATASOURCE
dim count as Integer
ds = thisShape.DATASOURCE(1)
count = ds.ColCount()
trace count
```

See Also RowCount method

ColorEntry Method

ColorEntry Method

Returns an instance of the **ColorEntry** object, corresponding to a color from the color palette of the document, by its index in the color collection.

Applies to: Document object

Syntax
[[**Set**] *colorEntryRet* =] *object*.**ColorEntry** (*index*)

The **ColorEntry** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Document** object.
index	Required. An expression that returns a **Long** value. The index of the color in the color palette.
colorEntryRet	Optional. A **ColorEntry** type variable.

Remarks

If *index* is less than **1** or greater than the number of colors in the color palette of the document, the **ColorEntry** method returns **Nothing**. To find out the number of the colors in the color palette of the document, use the **ColorsNum** method.

See Also ColorsNum method, ColorEntry object

ColorProperty Method

ColorProperty Method

Returns a **Color** object that corresponds to the color of the specified property of the shape.

Applies to: Shape object

Syntax
[[**Let**] *colorPropertyRet* =] *object*.**ColorProperty**(*propTag* [, *num*[, *geom*]])

The **ColorProperty** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.

num	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.
colorPropertyRet	Optional. A **Color** type variable.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or by using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all valid property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

ColorsNum Method

ColorsNum Method

Returns the number of colors in the color table of the document.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**ColorsNum** ()

The **ColorsNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Document** object.
countRet	Optional. A **Long** type variable.

Remarks

This method always returns a value greater than or equal to **1**, because there's always at least one color in the color table of the document. By defalut the color table of a new document contains **256** colors.

See Also ColorEntry method, ColorEntry object

ConnectDotsNum Method

ConnectDotsNum Method

Returns the number of connection points of the shape.

Applies to: Shape object

Syntax

[[**Let**] *countRet* =] *object*.**ConnectDotsNum** ()

The **ConnectDotsNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
countRet	Optional. A **Long** type variable.

Remarks

If the shape has no connection points, the **ConnectDotsNum** method returns **0**.

See Also AddConnectDot method, ConnectDot method, RemoveConnectDot method, ConnectDot object

ConnectDot Method

ConnectDot Method

Returns a **ConnectDot** object that corresponds to a connection point by its index in the connection point collection of the shape.

Applies to: Shape object

Syntax

[[**Set**] *connectDotRet* =] *object*.**ConnectDot** (*index*)

The **ConnectDot** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the shape's connection point.
connectDotRet	Optional. A **ConnectDot** type variable.

Remarks

If *index* is less than **1** or greater than the number of connection points of the *object* shape, the **ConnectDot** method returns **Nothing**. To find out the number of connection points of the shape, use the **ConnectDotsNum** method.

See Also AddConnectDot method, ConnectDotsNum method, RemoveConnectDot method, ConnectDot object

ControlDotsNum Method

ControlDotsNum Method

Returns the number of control handles that belong to the shape.

Applies to: Shape object

Syntax

[[**Let**] *countRet* =] *object*.**ControlDotsNum** ()

The **ControlDotsNum** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
countRet	Optional. A **Long** type variable.

Remarks

If *object* has no control handles, the **ControlDotsNum** method returns **0**.

See Also	AddControlDot method, ControlDot method, RemoveControlDot method, ControlDot object

ControlDot Method

ControlDot Method

Returns a **ControlDot** object that corresponds to a shape's control handle by its index in the control handle collection of the shape.

Applies to: Shape object

Syntax

[[**Set**] *controlDotRet* =] *object*.**ControlDot** (*index*)

The **ControlDot** method syntax has these Elements:

Element	Description

object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the control handle in the control handle collection of the shape.
controlDotRet	Optional. A **ControlDot** type variable.

Remarks

If *index* is less than **1** or greater than the number of control handles of the shape, the **ControlDot** method returns **Nothing**. To find out the number of control handles of the shape, use the **ControlDotsNum** method.

See Also	AddControlDot method, ControlDotsNum method, RemoveControlDot method, ControlDot object

ConvertToGroup Method

ConvertToGroup Method

Converts a shape from ConceptDraw vector picture format to a group of ConceptDraw shapes, preserving its location in the document.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**ConvertToGroup** (*shapeID*)

The **ConvertToGroup** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
shapeID	Required. An expression that returns a **Long** value. The ID of the shape to be converted.
shapeRet	Optional. A **Shape** type variable. The group of shapes resulting after the conversion.

Remarks

If the shape with the specified *shapeID* is not found in the shape collection of *object*, or the found shape is not a ConceptDraw vector picture, the **ConvertToGroup** method doesn't perform conversion and returns **Nothing**.

See Also ConvertToVFPicture method

ConvertToVFPicture Method

ConvertToVFPicture Method

Converts a ConceptDraw shape to a ConceptDraw vector picture, preserving its location in the document.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**ConvertToVFPicture** (*shapeID*)

The **ConvertToVFPicture** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
shapeID	Required. An expression that returns a **Long** value. The ID of the shape to be converted.
shapeRet	Optional. A **Shape** type variable. The ConceptDraw vector picture resulting after the conversion.

Remarks

If the shape with the specified *shapeID* is not found in the shape collection of *object*, the **ConvertToVFPicture** method doesn't perform conversion and returns **Nothing**. The method also returns **Nothing** when the shape with the specified ID can't be converted to a vector picture (for instance, if it's already a vector picture).

See Also ConvertToGroup method

CreateNewDoc Method

CreateNewDoc Method

Creates a new document and makes it active. Returns an instance of the **Document** object which corresponds to the created document.

Applies to: Application object

Syntax

[[**Set**] *documentRet* =] *object*.**CreateNewDoc** ()

The **CreateNewDoc** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the **Application** object.
documentRet	Optional. A **Document** type variable.

Remarks

A new document is created based on the default settings or from a template file. It's added to the end of the document collection of the application. So, the expression below returns the most recent created or opened document:

thisApp.Doc(thisApp.DocsNum()) ' returns most recent created or opened document

Example

This example contains an application-level script. It demonstrates using the **CreateNewDoc** method. The script creates a new document, which contains the shape with "New Document" text on the first page.

```
Dim newDoc as Document              ' Declare variables
Dim shp_rect As Shape
Set newDoc = thisApp.CreateNewDoc() ' Create a new document
newDoc.PageSizeX = 700     ' Set page width for the document
newDoc.PageSizeY = 700     ' Set page height for the document
' Draw a rectangle with "New Document" text
```

```
Set shp_rect = newDoc.Page(1).DrawRect( 50,50, newDoc.PageSizeX-50,
newDoc.PageSizeY-50 )
Set shp_rect.Text = "New Document"
```

See Also CloseDoc method, Doc method, DocByName method, DocsNum method, FirstDoc method, NextDoc method, OpenDoc method, Document object

CreateNewLib Method

CreateNewLib Method

Creates a new library and makes it active. Returns an instance of the **Library** object, corresponding to the created library.

Applies to: Application object

Syntax

[[**Set**] *libraryRet* =] *object*.**CreateNewLib** ()

The **CreateNewLib** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the **Application** object.
libraryRet	Optional. A **Library** type variable.

Remarks

A new library created with the **CreateNewLib** method becomes active and is added to the current library window. Note, that the new library is added to the end of the library collection of the document. That is, the following expression will return an instance of the **Library** object corresponding to the most recent created or open library:

thisApp.Lib(thisApp.LibsNum())

Example

This example contains an application-level script. The script creates a library and adds three shapes into it: a square, a circle and a triangle. The shapes are drawn in a temporary document, which is then closed without saving. The new library is saved in the current folder.

```
Dim newLib As Library           ' Declare variables
Dim tmpDoc As Document
Dim workPage As Page
Set newLib = thisApp.CreateNewLib()                   ' Create new library
newLib.Title= "Simple_Items"                          ' Choose  title
newLib.Name = "Simple_Items.cdl"                      ' and filename
Set tmpDoc = thisApp.CreateNewDoc()                   ' Create temporary document
Set workPage = tmpDoc.Page(1)                         ' Get reference to the
document page
workPage.DrawRect(0, 0, 400, 400).Text = "Square"     ' Draw square
workPage.DrawOval(0, 0, 400, 400).Text = "Circle"     ' Draw rectangle
workPage.BeginShape()                                 ' Draw triangle
workPage.MoveTo(0, 400)
workPage.LineTo(400, 400)
workPage.LineTo(200, 400 - 400*cos(3.1419/6) )
workPage.LineTo(0, 400)
workPage.EndShape().Text = "Triangle"
For i=1 to 3                                     ' Add the figures
    newLib.AddMaster( workPage.Shape(i) )        ' to newLib library
Next i
thisApp.CloseDoc( tmpDoc )  ' Close document without saving
newLib.Save()               ' Save library in current folder
```

See Also	CloseLib method, CreateNewLib method, FindLib method, Lib method, LibByName method, LibsNum method, OpenLib method, Library object

CSVColorValue Method

CSVColorValue Method

Returns an instance of **Color,** which contains information about the color, the value of which are located at the specified position in the table view a CSV file of the specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *color* =] *object.* **CSVColorValue** *(dsIndex, row, col)*

The **CSVColorValue** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.

dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view a CSV file of the specified data source object (shape).
col	Required. An expression that returns a **Long** value. The column number for the address in the table view a CSV file of the specified data source object (shape).
ret	Optional. A **Color** type variable.

Remarks

Line numbering and stobtsov in the table view CSV file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1.
Translated version of Example.docx

Example

Getting the color value, which is located on the second line in the third column, the second source of data in the collection of data source object (shape).

```
dim res as Color
res = thisShape.CSVColorValue (2,2,3)
if res.isRGB then
trace res.Red
trace res.Green
trace res.Blue
endif
```

See Also DataSource object , Color object , CSVText , CSVTextForKey , CSVValue ,CSVValueD , CSVValueDForKey , CSVValueForKey ,

CSVGetColumnForKey Method

CSVGetColumnForKey Method

Returns the column number, found by searching on a key in a table view of this CSV file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **CSVGetColumnForKey** *(dsIndex, keyRow, keyStr)*

The **CSVGetColumnForKey** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
keyRow	Required. An expression that returns a **Long** value. The line number with the key word for addressing a tabular representation of a CSV file of the specified data source object (shape).
keyStr	Required. An expression that returns a **String** value. Keyword search.
ret	Optional. A **Long** type variable.

Remarks

Line numbering in the table view CSV file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. In case of addressing the range of the table, or in the absence of data, or if the keyword is not found, returns 0.

Example

Getting the column number in the third row of the second source of data in the collection of data source object (shape). Find the column is the keyword "black".

```
dim res as Long
res = thisShape.CSVGetColumnForKey (2,3, "black")
trace res
```

See Also DataSource object , CSVText , CSVTextForKey , CSVValue , CSVValueD ,CSVValueDForKey , CSVValueForKey ,

CSVMinRowLength Method

CSVMinRowLength Method

Returns the minimum number of lines (from all the rows) in a tabular representation of a CSV file of the specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *length* =] *object.* **CSVMinRowLength** *(dsIndex)*

The **CSVMinRowLength** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
length	Optional. A **Long** type variable.

Remarks

The numbering of the data sources in the collection of data sources, the object starts at 1.

Example

Obtaining the minimum number of line items (of all lines) from the first data source in the collection of data sources, the object (shape).

```
dim num as Integer

num = thisShape.CSVMinRowLength (1)

trace num
```

See Also DataSource object , CSVRowMaxElement, CSVRowLength , CSVRowMinElement, CSVRowNum Method

CSVRowLength Method

CSVRowLength Method

Returns the number of elements in the specified row in a table view of this CSV file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *num* =] *object*. **CSVRowLength** *(dsIndex, row)*

The **CSVRowLength** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view a CSV file of the specified data source object (shape).
num	Optional. A **Long** type variable.

Remarks

Line numbering in the table view CSV file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1.

Example

Getting the number of elements of the last line of the first data source in the collection of data sources, the object (shape).

```
dim num as Integer
num = thisShape.CSVRowNum (1)
trace num
num = thisShape.CSVRowLength (1, num)
trace num
```

See Also DataSource object , CSVRowMaxElement, CSVMinRowLength ,CSVRowMinElement , CSVRowNum Method

CSVRowMaxElement Method

CSVRowMaxElement Method

Returns the minimum element of the specified row in a table view of this CSV file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **CSVRowMaxElement** *(dsIndex, row, defVal)*

The **CSVRowMaxElement** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view a CSV file of the specified data source object (shape).
defVal	Required. An expression that returns a **Double** value. The default value.
ret	Optional. A **Double** type variable.

Remarks

Line numbering in the table view CSV file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the maximum element of the second row from the first source of data sources in the collection of data object (shape).

```
dim res as Double
res = thisShape.CSVRowMaxElement (1,2, -1.5)
trace res
```

See Also DataSource object , CSVRowMinElement , CSVRowLength , CSVMinRowLength ,CSVRowNum Method

CSVRowMinElement Method

CSVRowMinElement Method

Returns the minimum element of the specified row in a table view of this CSV file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **CSVRowMinElement** *(dsIndex, row, defVal)*

The **CSVRowMinElement** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view a CSV file of the specified data source object (shape).
defVal	Required. An expression that returns a **Double** value.Znachenie by default.
ret	Optional. A **Double** type variable.

Remarks

Line numbering in the table view CSV file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting a minimal element of the second row from the first data source in the collection of data sources, the object (shape).

```
dim res as Double
res = thisShape.CSVRowMinElement (1,2, -1.5)
trace res
```

See Also DataSource object , CSVRowMaxElement , CSVRowLength , CSVMinRowLength, CSVRowNum Method

CSVRowNum Method

CSVRowNum Method

Returns the number of rows in a table view of this CSV file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *num* =] *object*. **CSVRowNum** *(index)*

The **CSVRowNum** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index in the collection of the data source data source object (shape).
num	Optional. A **Long** type variable.

Remarks

CSVRowNum method returns the number of non-empty rows in a table view CSV file data source object (shape). The numbering of the data sources in the collection of data sources, the object starts at 1.

Example

Getting the number of non-empty rows in a table view CSV file first data source in the collection of data source object (shape).

```
dim num as Integer
num = thisShape.CSVRowNum (1)
trace num
```

See Also DataSource object , CSVRowMaxElement, CSVMinRowLength , CSVRowLength Method , CSVRowMinElement

CSVTextForKey Method

CSVTextForKey Method

Returns the text found by searching on a key in a table view of this CSV file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **CSVTextForKey** *(dsIndex, keyRow, keyStr, valueRow, defVal)*

The **CSVTextForKey** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
keyRow	Required. An expression that returns a **Long** value. The line number with the key word for addressing a tabular representation of a CSV file of the specified data source object (shape).
keyStr	Required. An expression that returns a **String** value. Keyword search.
valueRow	Required. An expression that returns a **Long** value. The line number of the desired value for the address in the table view a CSV file of the specified data source object (shape).
defVal	Required. An expression that returns a **String** value. The default value.
ret	Optional. A **String** type variable.

Remarks

Line numbering in the table view CSV file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type, or if the keyword is not found.

Example

Getting the text, which are in the third line of the second source of data in the collection of data source object (shape). Find the column is the keyword "find", which is located on the second line of the source.

```
dim res as String
res = thisShape.CSVTextForKey (2,2, "find", 3, "Error")
trace res
```

See Also DataSource object , CSVText , CSVValue , CSVValueD , CSVValueDForKey ,CSVValueForKey , CSVValueType

CSVText Method

CSVText Method

Returns the text that are in the specified position in the table view a CSV file of the specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **CSVText** *(dsIndex, row, col, defVal)*

The **CSVText** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view a CSV file of the specified data source object (shape).
col	Required. An expression that returns a **Long** value. The column number for the address in the table view a CSV file of the specified data source object (shape).
defVal	Required. An expression that returns a **String** value. The default value.
ret	Optional. A **String** type variable.

Remarks

Line numbering and stobtsov in the table view CSV file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the string data, which are on the second line in the third column, the second source of data in the collection of data source object (shape).

```
dim res as String
res = thisShape.CSVText (2,2,3, "Data Error")
trace res
```

See Also DataSource object , CSVTextForKey , CSVValue , CSVValueD ,CSVValueDForKey , CSVValueForKey , CSVValue

CSVValueDForKey Method

CSVValueDForKey Method

Returns the value found using the search key in a table view of this CSV file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object*. **CSVValueDForKey** *(dsIndex, keyRow, keyStr, valueRow, defVal)*

The **CSVValueDForKey** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
keyRow	Required. An expression that returns a **Long** value. The line number with the key word for addressing a tabular representation of a CSV file of the specified data source object (shape).
keyStr	Required. An expression that returns a **String** value. Keyword search.

valueRow	Required. An expression that returns a **Long** value. The line number of the desired value for the address in the table view a CSV file of the specified data source object (shape).
defVal	Required. An expression that returns a **Double** value. The default value.
ret	Optional. A **Double** type variable.

Remarks

Line numbering in the table view CSV file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type, or if the keyword is not found.

Example

Getting the data, which are the first line of the second source of data in the collection of data source object (shape). Find the column is the keyword "height", which is located on the second line of the source.

```
dim res as Double
res = thisShape.CSVValueDForKey (2,2, "height", 1, -1.5)
trace res
```

See Also DataSource object , CSVText , CSVTextForKey , CSVValue , CSVValueD ,CSVValueForKey , CSVValueType

CSVValueD Method

CSVValueD Method

Gets a value that is in the specified position in the table view of this CSV file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **CSVValueD** *(dsIndex, row, col, defVal)*

The **CSVValueD** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.

dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view a CSV file of the specified data source object (shape).
col	Required. An expression that returns a **Long** value. The column number for the address in the table view a CSV file of the specified data source object (shape).
defVal	Required. An expression that returns a **Double** value. The default value.
ret	Optional. A **Double** type variable.

Remarks

Line numbering and stobtsov in the table view CSV file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the data that resides on the second line in the third column, the second source of data in the collection of data source object (shape).

```
dim res as Double
res = thisShape.CSVValueD (2,2,3, -1.5)
trace res
```

See Also DataSource object , CSVText , CSVTextForKey , CSVValue , CSVValueDForKey, CSVValueForKey , CSVValueTy

CSVValueForKey Method

CSVValueForKey Method

Returns the integer value found by searching on a key in a table view of this CSV file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **CSVValueForKey** *(dsIndex, keyRow, keyStr, valueRow, defVal)*

The **CSVValueForKey** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
keyRow	Required. An expression that returns a **Long** value. The line number with the key word for addressing a tabular representation of a CSV file of the specified data source object (shape).
keyStr	Required. An expression that returns a **String** value. Keyword search.
valueRow	Required. An expression that returns a **Long** value. The line number of the desired value for the address in the table view a CSV file of the specified data source object (shape).
defVal	Required. An expression that returns a **Long** value. The default value.
ret	Optional. A **Long** type variable.

Remarks

Line numbering in the table view CSV file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type, or if the keyword is not found.

Example

Getting the data that resides in the third row of the second source of data in the collection of data source object (shape). Find the column is the keyword "black", which is located on the second line of the source.

```
dim res as Long
res = thisShape.CSVValueForKey (2,2, "black", 3, -1)
trace res
```

See Also DataSource object , CSVText , CSVTextForKey , CSVValue , CSVValueD ,CSVValueDForKey , CSVValueType

CSVValueType Method

CSVValueType Method

Returns the type of data that resides in the specified position in the table view a CSV file of the specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object*. **CSVValueType** *(dsIndex, row, col)*

The **CSVValueType** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view a CSV file of the specified data source object (shape).
col	Required. An expression that returns a **Long** value. The column number for the address in the table view a CSV file of the specified data source object (shape).
ret	Optional. A **Long** type variable.

Remarks

Line numbering and stobtsov in the table view CSV file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. Interpretation of the return value: 0 - Void; 1 - String; 2 - Integer; 3 - Float; 4 - Color; 5 - Date;

Example

Getting the data type, which are located on the second line in the third column, the second source of data in the collection of data source object (shape).

```
dim res as Long
res = thisShape.CSVValueType (2,2,3)
trace res
```

See Also DataSource object , CSVText , CSVTextForKey , CSVValue , CSVValueD ,CSVValueDForKey , CSVValueForKey

CSVValue Method

CSVValue Method

Returns an integer value that is at the specified position in the table view of this CSV file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **CSVValue** *(dsIndex, row, col, defVal)*

The **CSVValue** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view a CSV file of the specified data source object (shape).
col	Required. An expression that returns a **Long** value. The column number for the address in the table view a CSV file of the specified data source object (shape).
defVal	Required. An expression that returns a **Long** value. The default value.
ret	Optional. A **Long** type variable.

Remarks

Line numbering and stobtsov in the table view CSV file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the data that resides on the second line in the third column, the second source of data in the collection of data source object (shape).

```
dim res as Long
res = thisShape.CSVValue (2,2,3, -1)
trace res
```

See Also DataSource object , CSVText , CSVTextForKey , CSVValueD ,CSVValueDForKey , CSVValueForKey , CSVValueT

CustomPropByLabel Method

CustomPropByLabel Method

Returns an instance of the appropriate signature **CustomProp** custom property in the collection of custom properties of the object (shape).

Applies to: Shape object

Syntax

[[Set] *customPropRet* =] *object.* **CustomPropByLabel** *(label)*

The **CustomPropByLabel** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
label	Required. An expression that returns a **String** value. Label of the custom property in the custom property collection of the shape.
customPropRet	Optional. A **CustomProp** type variable.

Remarks

If the custom property with such a signature **label is** not found, the **CustomPropByLabel** method returns **Nothing.** To find out the number of custom properties of the shape, use the **CustomPropsNum**method.

Example

Getting a custom property from the collection of custom properties of the object (shape), whose signature matches the signature of "Shape Label".

```
dim resCusProp as CustomProp
resCusProp = thisShape.CustomPropByLabel ("Shape Label")
if (resCusProp <> NULL) then
trace resCusProp.Prompt
else
trace "NULL"
end if
```

See Also AddCustomProp Method , Method CustomProp , CustomPropsNum Method ,Method RemoveCustomProp, CustomProp object

CustomPropsNum Method

CustomPropsNum Method

Returns the number of custom properties of the shape.

Applies to: Shape object

Syntax

[[Let] *countRet* =] *object.* **CustomPropsNum** ()

The **CustomPropsNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
countRet	Optional. A **Long** type variable.

Remarks

If the shape has no custom properties, the **CustomPropsNum** method returns **0.**

See Also AddCustomProp Method , Method CustomProp , CustomPropByLabel ,RemoveCustomProp Method, CustomProp object

CustomProp Method

CustomProp Method

Returns a **CustomProp** object that corresponds to a custom property by its index in the custom property collection of the shape.

Applies to: Shape object

Syntax

[[Set] *customPropRet* =] *object.* **CustomProp** *(index)*

The **CustomProp** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
index	Required. An expression that returns a **Long** value. The index of the custom property in the custom property collection of the shape.
customPropRet	Optional. A **CustomProp** type variable.

Remarks

If *index* is less than **1** or greater than the number of custom properties of the shape, the **CustomProp** method returns **Nothing.** To find out the number of custom properties of the shape, use the**CustomPropsNum** method.

See Also AddCustomProp Method , CustomPropByLabel , CustomPropsNum Method ,Method RemoveCustomProp, CustomProp object

DataSourcesNum Method

DataSourcesNum Method

Returns the number of data sources in the object (shape).

Applies to: Shape object

Syntax

[[Let] *num* =] *object*. **DataSourcesNum** ()

The **DataSourcesNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
num	Optional. A **Long** type variable.

Remarks

DataSourcesNum method returns the total number of data sources, regardless of whether they are valid or not. If the object has no data sources, the function returns 0.

Example

```
dim num as Integer
num = thisShape.DataSourcesNum ()
trace num
```

See Also DataSource object , AddDataSource Method , Method DataSource , Method RemoveDataSource

DataSource Method

DataSource Method

Returns an instance of the **DataSource** object from the collection of data sources, the object (shape) of the index.

Applies to: Shape object

Syntax

[[**Set**] *dataSourceRet* =] *object* . **DataSource** (*index*)

The **DataSource** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the DataSource in the DataSources collection of the shape.
dataSourceRet	Optional. A **DataSource** type variable.

Remarks

AddDataSource method in case of failure returns 0.

The numbering of the indices of data sources in the collection begins with 1.

Example

```
dim num as Integer
dim ds as DataSource
num = thisShape.DataSourcesNum()
trace num
ds = thisShape.DataSource(num)
trace ds.DataSource
```

See Also	DataSource object , AddDataSource method , DataSourcesNum method ,RemoveDataSource method

DeflateRect Method

DeflateRect Method

"Shrinks" a rectangle by its X and Y axis, calculates new coordinates of the object.

Applies to objects: DRect

Syntax

object . **DeflateRect** (*x*, *y*)

The **DeflateRect** statement syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
x	A Double value, that specifies the offset for the right and left sides of the rectangle.
y	A Double value, that specifies the offset for the top and bottom sides of the rectangle.

Remarks

"Shrinking" a rectangle doesn't reposition its center. The following formulas are used to calculate the coordinates:

left = left + x; top = top + x; right = right - x; bottom = bottom - x

Example

```
' create an instance of the object
Dim MyObject as new DRect
' set left,top,right,bottom properties of object
MyObject.SetRect(200,200,1000,1000)
' shrink the rectangle
' After the operation the values will be as follows:
' left - 300, top - 300, right - 900, bottom - 900
MyObject.DeflateRect(100,100)
```

See Also

DRect Object , InflateRect Method

DeselectAll Method

DeselectAll Method

Removes selection from all shapes that belong to the page or group displayed in the window.

Applies to: Window object

Syntax

[[**Let**] *boolRet* =] *object* . **DeselectAll** ()

The **DeselectAll** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Window** object.
boolRet	Optional. A **Boolean** type variable.

Remarks

This method is only effective when the window is of the document view type (see the **Type** property). For windows of other type the **DeselectAll** method returns **False** .

The **DeselectAll** method removes selection from all shapes displayed in the active page or group window and returns **True** . If there are no selected shapes in the window, it returns **False** . An inverse method to **DeselectAll** is the **SelectAll** method, which selects all shapes in the window.

See Also Type property , Deselect method , GetSelectedService method , GetSelectedShape method , Select method , SelectAll method , SelectedNum method

Deselect Method

Deselect Method

Removes selection from a shape with the specified **ID** , that belongs to the page or group displayed in the window.

Applies to: Window object

Syntax

[[**Let**] *boolRet* =] *object* . **Deselect** (*shapeID*)

The **Deselect** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Window** object.
shapeID	Required. An expression that returns a **Long** value. An ID of the shape to be deselected.
boolRet	Optional. A **Boolean** type variable.

Remarks

This method is only effective when the window is of the library view type (see the **Type** property). For windows of other type the **Deselect** method returns **False** .

If there is the shape with the specified ID among the shapes displayed in the page or group window, the method deselects the shape and returns **True** , otherwise it returns **False** . An inverse method to**Deselect** is the **Select** method, which selects a shape with the specified ID.

See Also ID property , Type property , DeselectAll method , GetSelectedService method ,GetSelectedShape method , Select method , SelectAll method , SelectedNum method

DocByName Method

DocByName Method

Searches for a document with the specified name (**Name** property) among the open documents of the application. Returns an instance of the **Document** object corresponding to the found document.

Applies to: Application object

Syntax

[[**Set**] *documentRet* =] *object* . **DocByName** (*docName*)

The **DocByName** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the **Application** object.
docName	Required. An expression that returns a **String** value. The name (**Name** property) of the document being searched.
documentRet	Optional. A **Document** type variable.

Remarks

The **DocByName** method searches for a document with the *docName* name starting from the first document in the document collection and returns the first found document. That is, if the third and fifth document have the same name, the **DocByName** method will returns the instance of the **Document** object that corresponds to the third document. If there is no matching document, the method returns **Nothing**.

Example

This example contains an application-level script. The program first askes the user to enter the name of the document and then searches for the the document with the provided name. If the search is successfu, it maximizes the active window of the found document.

```
' Declare variables
```

```
Dim inStr As String
Dim resDoc As Document
' Show the dialog where to input the document name
Set inStr = InputBox( "Enter document name:", "Document by name!",
"Concept1.cdd" )
' Find the specified document
Set resDoc = thisApp.DocByName( inStr )
' If the document is found, activate it
If resDoc <> Nothing Then
resDoc.ActiveView.Maximize()
' Otherwise inform the user
Else
MsgBox( "The document " & inStr & " is not found!" )
End If
```

See Also Name property , CloseDoc method , CreateNewDoc method , Doc method ,DocsNum method , FirstDoc method , NextDoc method , OpenDoc method ,Document object

DocsNum Method

DocsNum Method

Returns the number of open documents in the application.

Applies to: Application object

Syntax

[[**Set**] *countRet* =] *object* . **DocsNum** ()

The **DocsNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Application** object.
countRet	Optional. A **Long** type variable.

Remarks

It's convenient to use the **DocsNum** method together with the **Doc** method to go through the open documents in the application.

Example

This example contains an application-level script. It displays the number of windows for each of the documents open in the application, the number of the open documents and the total number of document views in the application.

```
Dim curDoc As Document ' Declare variables
Dim resStr As String
Dim vcount As Integer
TRACE "-----------------------------------------"
vcount = 0
For i=1 To thisApp.DocsNum() ' Loop through all documents
Set curDoc = thisApp.Doc(i) ' Get next document
TRACE curDoc.Name & " -- " & curDoc.ViewsNum() ' Display the number of views
of each document
vcount = vcount + curDoc.ViewsNum()
Next i
' Display the number of documents
TRACE "Count of documens = " & thisApp.DocsNum
TRACE "Count of views = " & vcount ' Display the number of document windows
TRACE "-----------------------------------------"
```

See Also	CloseDoc method , CreateNewDoc method , Doc method , DocByName method ,FirstDoc method , NextDoc method , OpenDoc method , Document object

Doc Method

Doc Method

Returns an instance of the **Document** object by its index in the document collection of the application.

Applies to: Application object

Syntax

[[**Set**] *documentRet* =] *object*.**Doc** (*index*)

The **Doc** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Application** object.
index	Required. An expression that returns a **Long** value. Represents the index of the document in the document collection of the application.
documentRet	Optional. A **Document** type variable.

Remarks

If *index* is less than **1** or greater than the number of open documents the **Doc** method returns **Nothing**. To find out the number of open documents, use the **DocsNum** method.

When creating a new document or opening an existing one the document is added to the end of the list of the open documents. That is, the index of the most recent open document is equal to the number of the oepn document.

Example

This example contains an application-level script. It demonstrates how to export all open documents to the PNG format. The example uses the **Doc** method to go through all open documents.

```
Dim curDoc As Document              ' Declare a document variable
For i=1 To thisApp.DocsNum()        ' Loop through all open documents
    Set curDoc = thisApp.Doc(i)     ' Get the i document
    If curDoc.Export( curDoc.Name & ".png", 0, False, False ) Then' Export
document to png format
        TRACE curDoc.Name & ".png"      ' Display name of the file
    End If
Next i
```

See Also CloseDoc method, CreateNewDoc method, DocByName method, DocsNum method, FirstDoc method, NextDoc method, OpenDoc method, Document object

DoForConnected Method

DoForConnected Method

Causes BASIC procedure with an appropriate title for each of the objects (shapes), connected (directly or through other objects) to the object (shape) with the specified identifier.

Applies to: Page object

Syntax

object.**DoForConnected** (*funcname*, *id*)

The **DoForConnected** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an **Page** object.

funcname	Required. An expression that returns Value of the **String type**. The name of the BASIC procedure to be called.
id	Required. An expression that returns Value of the **Double type**. ID of the object to which append objects to the called procedure.

Remarks

I append the object to be compiled and run BASIC script.

Example

This example contains a shape-level script. Implementation of this procedure to execute the BASIC function called "Add" to all objects (shapes), a BASIC script, which it will be found and connected (directly or through other objects) to the object (shape) with an ID of 11.

thisPage.DoForConnected ("Add", 11)

See Also Shape object, Page object, Document object

DrawConnector Method

DrawConnector Method

Draws a connector. Returns an instance of the **Shape** object that represents the created shape.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**DrawConnector** (*xBegin*, *yBegin*, *xEnd*, *yEnd*)

The **DrawConnector** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object from the **Applies to** list.
xBegin	Required. A **Double** value that represents the X-coordinate of the connector's begin point.
yBegin	Required. A **Double** value that represents the Y-coordinate of the connector's begin point.

xEnd	Required. A **Double** value that represents the X-coordinate of the connector's end point.
yEnd	Required. A **Double** value that represents the Y-coordinate of the connector's end point.
shapeRet	Optional. A **Shape** type variable.

Remarks

If *object* is a page or a group, the **DrawConnector** method creates the new connector inside this group / page, and then tries to connect the created connector. If the endpoints of the connector coincide with the default or user-defined connection points on any shapes, the connector is connected to these points. For shapes of other types the **DrawConnector** method doesn't build anything and always returns **Nothing**.

The coordinates of the points are specified in the coordinate system of the shape, group or page, *object* is associated with. The coordinates are measured in **InternalUnit**s.

See Also DrawSmartConnector method

DrawGroup Method

DrawGroup Method

Creates a group with the specified position, width and height. Returns an instance of the **Shape** object representing this group.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**DrawGroup** (*xLeft*, *yTop*, *xRight*, *yBottom*)

The **DrawGroup** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object from the **Applies to** list.

xLeft	Required. A **Double** value that represents the X-coordinate of the top left corner of the rectangle that is the group's bound.
yTop	Required. A **Double** value that represents the Y-coordinate of the top left corner of the rectangle that is the group's bound.
xRight	Required. A **Double** value that represents the X-coordinate of the bottom right corner of the rectangle that is the group's bound.
yBottom	Required. A **Double** value that represents the Y-coordinate of the bottom right corner of the rectangle that is the group's bound.
shapeRet	Optional. A **Shape** type variable.

Remarks

If *object* is a page or a group, the **DrawGroup** method creates a new group with the specified dimensions and adds it to the shape collection of the corresponding page or group. The coordinate origin of the created group (the **GPinX** and **GPinY** properties) is set in the point with the *xLeft* and *yTop* coordinates. The new group contains no shapes.

If *object* is a simple shape (not group), the **DrawGroup** method takes no action and returns **Nothing**.

The coordinates of the points are specified in the coordinate system of the shape, group or page to which the instance of the *object* object corresponds. The coordinates are set in internal units (**InternalUnit**).

See Also GPinX property, GPinY property

DrawGuide Method

DrawGuide Method

Draws a guide line based on the specified coordinates of the coordinate origin (the **GPinX** and **GPinY** properties) of the guide line and the angle, to which it's turned with respect to its coordinate origin. Returns an instance of the **Shape** object that represents the created service object.

Applies to: Page object, Shape object

Syntax

[[**Set**] *servObjRet* =] *object*.**DrawGuide** (*xGPin*, *yGPin*, *angle*)

The **DrawGuide** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object from the **Applies to** list.
xGPin	Required. An expression that returns a **Double** value. Represents the X-coordinate of the coordinate origin (the **GPinX** property).
yGPin	Required. An expression that returns a **Double** value. Represents the Y-coordinate of the coordinate origin (the **GPinY** property).
angle	Required. An expression that returns a **Double** value. Represents the angle to which the guide line is turned counter-clockwise relevant to its horizontal position in the coordinate system of *object*.
servObjRet	Optional. A **Shape** type variable.

Remarks

When *object* is a page or a group, the **DrawGuide** method creates a guide line and adds it to the service object collection of the corresponding page or group. The coordinates of the origin of the guide are specified in the coordinate system of *object*. The *angle* value is set in radians.

If *object* is a simple shape, the **DrawGuide** method takes no action and returns **Nothing**.

See Also	RemoveServObj method, RemoveServObjByID method, ReorderServObj method, ReorderServObjByID method, ServObj method, ServObjByID method, ServObjsNum method, ServObj object

DrawLine Method

DrawLine Method

Draws a line. Returns an instance of the **Shape** object that corresponds to the created shape.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**DrawLine** (*xBegin*, *yBegin*, *xEnd*, *yEnd*)

The **DrawLine** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
xBegin	Required. An expression that returns a **Double** value, representing the X-coordinate of the line's begin point.
yBegin	Required. An expression that returns a **Double** value, representing the Y-coordinate of the line's begin point.
xEnd	Required. An expression that returns a **Double** value, representing the X-coordinate of the line's end point.
yEnd	Required. An expression that returns a **Double** value, representing the Y-coordinate of the line's end point.
shapeRet	Optional. A **Shape** type variable.

Remarks

If *object* is a page or a group, the **DrawLine** method draws the shape on the page or in the group. In the shape it adds a new geometry that describes the line with the specified coordinates. Then it returns an instance of the **Shape** object that corresponds to the new shape. If **DrawLine** was called after the **BeginShape** method, it adds a new geometry to the current Basic shape and then returns an instance of the **Shape** object, corresponding to that shape.

If *object* is a simple shape (not a group), the **DrawLine** method draws a line in this shape and returns*object*.

The coordinates of the points are specified in the coordinate system of the shape, group or page to which the instance of *object* corresponds. The unit of measure for the coordinates are the internal units (**InternalUnit**).

See Also BeginShape method, DrawOval method, DrawRect method, DrawSector method, EndShape method, LineTo method

DrawOval Method

DrawOval Method

Draws an ellipse. Returns an instance of the **Shape** object which corresponds to the drawn shape or the shape in which the ellipse was built.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**DrawOval** (*xLeft*, *yTop*, *xRight*, *yBottom*)

The **DrawOval** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
xLeft	Required. An expression that returns a **Double** value. It represents the X-coordinate of the top left corner of the rectangle circumscribing the ellipse.
yTop	Required. An expression that returns a **Double** value. It represents the Y-coordinate of the top left corner of the rectangle circumscribing the ellipse.
xRight	Required. An expression that returns a **Double** value. It represents the X-coordinate of the bottom right corner of the rectangle circumscribing the ellipse.
yBottom	Required. An expression that returns a **Double** value. It represents the Y-coordinate of the bottom right corner of the rectangle circumscribing the ellipse.
shapeRet	Optional. A **Shape** type variable.

Remarks

The ellipse is drawn by the specified coordinates of the rectangle, circumscribing the ellipse. The sides of the rectangle are equal to the diameters of the ellipse.

If *object* is a page or a group, the **DrawOval** method creates on that page or group a shape containing a geometry that describes the ellipse with the specified size and coordinates. Then it returns an instance of the **Shape** object, that corresponds to the created shape. If the **DrawOval** method was called after the **BeginShape** method, it adds a geometry describing the ellipse to the current Basic shape. Then it returns an instance of the **Shape** object, that corresponds to the current Basic shape of the page or group.

If *object* is a simple shape, the **DrawOval** method draws the ellipse in this shape and returns *object*.

The coordinates of the points are specified in the coordinate system of the shape, group or page to which the instance of *object* corresponds. The coordinates are measured in the internal units (**InternalUnit**).

See Also BeginShape method, DrawLine method, DrawRect method, DrawSector method, EndShape method

DrawRect Method

DrawRect Method

Draws a rectangle. Returns an instance of the **Shape** object which corresponds to the drawn shape, or the shape in which the rectangle has been drawn.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**DrawRect** (*xLeft*, *yTop*, *xRight*, *yBottom*)

The **DrawRect** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object from the **Applies to** list.
xLeft	Required. An expression that returns a **Double** value. It represents the X-coordinate of the top left corner of the rectangle.
yTop	Required. An expression that returns a **Double** value. It represents the Y-coordinate of the top left corner of the rectangle.
xRight	Required. An expression that returns a **Double** value. It represents the X-coordinate of the bottom right corner of the rectangle.
yBottom	Required. An expression that returns a **Double** value. It represents the Y-coordinate of the bottom right corner of the rectangle.
shapeRet	Optional. A **Shape** type variable.

Remarks

If *object* is a page or a group, the **DrawRect** method creates on that page or group a shape containing a geometry that describes the rectangle with the specified size and coordinates. Then in returns an instance of the **Shape** object, that corresponds to the created shape. If the **DrawRect** method was called after the **BeginShape** method, it adds a geometry describing the rectangle to the current Basic shape. Then it returns an instance of the **Shape** object, that corresponds to the current Basic shape of the page or group.

If *object* is a simple shape, the **DrawRect** method draws the rectangle in this shape and returns *object*.

When using the **DrawRect** method, the order in which the coordinates are specified is not significant. The coordinates of the points are specified in the coordinate system of the shape, group or page to which the instance of *object* corresponds. The coordinates are measured in the internal units (**InternalUnit**).

See Also	BeginShape method, DrawLine method, DrawOval method, DrawSector method, EndShape method

DrawSector Method

DrawSector Method

Draws an arc of a circle. Returns an instance of the **Shape** object which corresponds to the drawn shape or the shape in which the arc has been built.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**DrawSector** (*xBegin*, *yBegin*, *xEnd*, *yEnd*, *xMiddle*, *yMiddle*)

The **DrawSector** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
xBegin	Required. An expression that returns a **Double** value. Represents the X-coordinate of the begin point of the arc being built.
yBegin	Required. An expression that returns a **Double** value. Represents the Y-coordinate of the begin point of the arc being built.
xEnd	Required. An expression that returns a **Double** value. Represents the X-coordinate of the end point of the arc being built.
yEnd	Required. An expression that returns a **Double** value. Represents the Y-coordinate of the end point of the arc being built.
xMiddle	Required. An expression that returns a **Double** value. Represents the X-coordinate of the point, that lies on the arc being built.

yMiddle	Required. An expression that returns a **Double** value. Represents the Y-coordinate of the point, that lies on the arc being built.
shapeRet	Optional. A **Shape** type variable.

Remarks

If *object* is a page or a group, the **DrawSector** method creates on the corresponding page or group a shape, that contains a geometry describing an arc of a circle with the specified coordinates of the begin, end points and the point that lies on the arc. Then it returns an instance of the **Shape** object, corresponding to that shape. If the **DrawSector** method was called after the **BeginShape** method, it adds a new geometry, describing the arc, to the current Basic-shape. Then it returns an instance of the **Shape** object, corresponding to that shape.

If *object* is a simple shape, the **DrawSector** method for the shape creates an arc in the shape and returns *object*.

The coordinates of the points are in the coordinate system of the shape, group or the page - depending on the *object* type. The unit of measure of the specified coordinates is **InternalUnit**.

See Also	BeginShape method, DrawLine method, DrawOval method, DrawRect method, EndShape method

DrawSmartConnector Method

DrawSmartConnector Method

Draws a smart connector. Returns a **Shape** object that corresponds to the created shape.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**DrawSmartConnector** (*xBegin*, *yBegin*, *xEnd*, *yEnd*)

The **DrawSmartConnector** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
xBegin	Required. An expression that returns a **Double** value. The X-coordinate of the smart connector's begin point.

yBegin	Required. An expression that returns a **Double** value. The Y-coordinate of the smart connector's begin point.
xEnd	Required. An expression that returns a **Double** value. The X-coordinate of the smart connector's end point.
yEnd	Required. An expression that returns a **Double** value. The Y-coordinate of the smart connector's end point.
shapeRet	Optional. A **Shape** type variable.

Remarks

If *object* is a page or a group, the **DrawSmartConnector** method creates the new smart connector inside this group / page, and then tries to connect the created smart connector. If the endpoints of the smart connector coincide with the default or user-defined connection points on any shapes, the connector is connected to these points. For shapes of other types the **DrawSmartConnector** method doesn't build anything and always returns **Nothing**.

The coordinates of the points are specified in the coordinate system of the shape, group or page, *object* is associated with. The coordinates are measured in internal units (**InternalUnit**).

See Also DrawConnector method

DrawStampSelection Method

DrawStampSelection Method

Creates on this page / in this group a copy of selected shapes, with the specified dimensions and position, similar to the Stamp Tool in ConceptDraw. Returns a **Shape** object that corresponds to the created shape.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**DrawStampSelection** (*xLeft*, *yTop*, *xRight*, *yBottom*)

The **DrawStampSelection** method syntax has these Elements:

Element	**Description**

object	Required. An expression that returns an object in the **Applies to** list.
xLeft	Required. An expression that returns a **Double** value. The X-coordinate of the top left corner of the bound that will encompass the copy of the selected shapes.
yTop	Required. An expression that returns a **Double** value. The Y-coordinate of the top left corner of the bound that will encompass the copy of the selected shapes.
xRight	Required. An expression that returns a **Double** value. The X-coordinate of the bottom right corner of the bound that will encompass the copy of the selected shapes.
yBottom	Required. An expression that returns a **Double** value. The Y-coordinate of the bottom right corner of the bound that will encompass the copy of the selected shapes.
shapeRet	Optional. A **Shape** type variable.

Remarks

Selected shapes are the shapes selected on the active page of the ConceptDraw document to which *object* belongs. If a copy of the selected shapes couldn't be created, the **DrawStampSelection** method returns **Nothing**. Copies of the selected shapes are grouped into one group, which is then positioned at the specified coordinates. If the operation has been successful, the method returns a **Shape** object that corresponds to the newly created shape or group.

Note, that the order in which the coordinates of the bound encompassing the copy of the shape are specified, is not significant. The coordinates are specified in the coordinate system of the shape, group or page to which the instance of *object* corresponds. The coordinates are measured in internal units (**InternalUnit**).

See Also DrawStamp method, DropStamp method, DropStampSelection method

DrawStamp Method

DrawStamp Method

Creates on this page / in this group a copy of the specified shape, with the specified dimensions and position, similar to the Stamp Tool in ConceptDraw. Returns a **Shape** object that corresponds to the created shape.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**DrawStamp** (*shapeToStamp*, *xLeft*, *yTop*, *xRight*, *yBottom*)

The **DrawStamp** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
shapeToStamp	Required. An expression that returns a **Shape** object. The shape to be copied.
xLeft	Required. An expression that returns a **Double** value. The X-coordinate of the top left corner of the bound that will encompass the copy of the shape in *shapeToStamp*.
yTop	Required. An expression that returns a **Double** value. The Y-coordinate of the top left corner of the bound that will encompass the copy of the shape in *shapeToStamp*.
xRight	Required. An expression that returns a **Double** value. The X-coordinate of the bottom right corner of the bound that will encompass the copy of the shape in *shapeToStamp*.
yBottom	Required. An expression that returns a **Double** value. The Y-coordinate of the bottom right corner of the bound that will encompass the copy of the shape in *shapeToStamp*.
shapeRet	Optional. A **Shape** type variable.

Remarks

If a copy of *shapeToStamp* couldn't be created, the **DrawStamp** method returns **Nothing**.

Note, that the order in which the coordinates of the bound encompassing the copy of the shape are specified, is not significant. The coordinates are specified in the coordinate system of the shape, group or page to which the instance of *object* corresponds. The coordinates are measured in internal units (**InternalUnit**).

See Also DrawStamp method, DrawStampSelection method, DropStamp method, DropStampSelection method

DropStampSelection Method

DropStampSelection Method

Creates on this page / in this group a copy of selected shapes, and places it to the specified position, similar to the Stamp Tool in ConceptDraw. Returns a **Shape** object that corresponds to the created shape.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**DropStampSelection** (*xGPin*, *yGPin*)

The **DropStampSelection** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
xGPin	Required. An expression that returns a **Double** value. The X-coordinate of the rotation center (the **GPinX** property) for the copy of the selected shapes.
yGPin	Required. An expression that returns a **Double** value. The Y-coordinate of the rotation center (the **GPinY** property) for the copy of the selected shapes.
shapeRet	Optional. A **Shape** type variable.

Remarks

Selected shapes are the shapes selected on the active page of the ConceptDraw document to which *object* belongs. If a copy of the selected shapes couldn't be created, the **DropStampSelection** method returns **Nothing**. Copies of the selected shapes are grouped into one group, which is then positioned at the specified coordinates. If the operation has been successful, the method returns a **Shape** object that corresponds to the newly created shape or group.

The coordinates are specified in the coordinate system of the shape, group or page to which the instance of *object* corresponds. The coordinates are measured in internal units (**InternalUnit**).

See Also	GPinX property, GPinY property, DrawStamp method, DrawStampSelection method, DropStamp method

DropStamp Method

DropStamp Method

Creates on this page / in this group a copy of the specified shape, and places it to the specified position, similar to the Stamp Tool in ConceptDraw. Returns a **Shape** object that corresponds to the created shape.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**DropStamp** (*shapeToStamp*, *xGPin*, *yGPin*)

The **DropStamp** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
shapeToStamp	Required. An expression that returns a **Shape** object. The shape to be copied.
xGPin	Required. An expression that returns a **Double** value. The X-coordinate of the rotation center (the **GPinX** property) for the copy of the shape in *shapeToStamp*.
yGPin	Required. An expression that returns a **Double** value. The Y-coordinate of the rotation center (the **GPinY** property) for the copy of the shape in *shapeToStamp*.
shapeRet	Optional. A **Shape** type variable.

Remarks

If a copy of *shapeToStamp* couldn't be created, the **DropStamp** method returns **Nothing**.

The coordinates are specified in the coordinate system of the shape, group or page to which the instance of *object* corresponds. The coordinates are measured in internal units (**InternalUnit**).

See Also GPinX property, GPinY property, DrawStamp method, DrawStampSelection method, DropStampSelection method

DSValueEl Method

Returns an instance of an object by name DataSourceValue line (field Name) Data Table parameters of the object (shape), containing in the Value data list.

Applies to: Shape object

Syntax

[[Set] *dataSourceValueRet* =] *object*. **DSValueEl** *(name, index)*

The **DSValueEl** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
name	Required. An expression that returns a **String** value. The data row of the Name field Data parameters of the object (shape), which in the Value field contains the value of interest.
index	Required. An expression that returns a **Long** value. Number of list item, contained in the Value row Data.
dataSourceValueRet	Optional. A **DataSourceValue** type variable.

Remarks

DSValueEl method in case of failure returns 0. The numbering of the list item, contained in the Value Data Table parameters of the object (shape) starts with 1. The list in the Value field is a set of values, separated by a comma.

Example

```
For example, Table Data object parameters (shape) has a row with a value in
the field Name - "first".
In the Value field of this line there is a list of values - "9,777.777,999.99,
Value El, 20."
As a result of performing a function in BASIC editor will print 777,777.
dim num as Integer
dim ds as DataSourceValue
ds = thisShape.DSValueEl ("first", 2)
trace ds.value
```

See Also	DataSourceValue object , AddDSValue Method , Method DSValue , DSValuesNum Method , Method RemoveDSValue

DSValuesNum Method

DSValuesNum Method

Returns the number of rows in a table Data parameters of the object (shape).

Applies to: Shape object

Syntax

[[Let] *num* =] *object*. **DSValuesNum** ()

The **DSValuesNum** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
num	Optional. A **Long** type variable.

Remarks

If the object does not have a table Data parameters of the object (shape), **DSValuesNum** method returns 0.

Example

```
dim num as Integer
num = thisShape.DSValuesNum ()
trace num
```

See Also	DataSourceValue object , AddDSValue Method , Method DSValue , DSValueEl Method , Method RemoveDSValue

DSValue Method

DSValue Method

Returns an instance of an object DataSourceValue, containing data from a table row Data parameters of the object (shape) of the index.

Applies to: Shape object

Syntax

[[Set] *dataSourceValueRet* =] *object*. **DSValue** *(index)*

The **DSValue** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the row Data parameters of the object (shape).
dataSourceValueRet	Optional. A **DataSourceValue** type variable.

Remarks

DSValue method in case of failure returns 0. The numbering of the indices of rows in a table Data begins at 1.

Example

```
dim num as Integer
dim ds as DataSourceValue
num = thisShape.DSValuesNum ()
trace num
ds = thisShape.DSValue (num)
trace ds.Value
```

See Also	DataSourceValue object, AddDSValue method, DSValueEl method, DSValuesNum method, RemoveDSValue method

EndRebuild Method

EndRebuild Method

Informs the ConceptDraw engine about the termination of modifying properties of the shapes of the document.

Applies to: Document object

Syntax

object.**EndRebuild** ()

The **EndRebuild** method syntax has these Elements:

Element	Description

object	Required. An expression, that returns a **Document** object.

Remarks

Calling this method must be preceded by calling the **StartRebuild** method, which informs the ConceptDraw engine about the start of modifying properties of the shapes of the document. This scheme of modifying shape properties is used when it's necessary to modify several properties of the shapes without re-calculating properties after each change. The properties are re-calculated only once after the **EndRebuild** method has been called.

See Also [StartRebuild method](), [UpdateAllViews method]()

EndShape Method

EndShape Method

Returns an instance of the **Shape** object which corresponds to the current Basic shape and informs ConceptDraw that the shape has been built.

Applies to: [Page object](), [Shape object]()

Syntax
[[**Set**] *shapeRet* =] *object*.**EndShape** ()

The **EndShape** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
currentBasicShapeRet	Optional. A **Shape** type variable.

Remarks

This method is only effective for pages and shape that are groups. For all other shapes this method returns **Nothing**. Calling **EndShape** must be preceded by **BeginShape**, which initialized the current Basic shape of the group/page. Otherwise the **EndShape** method returns **Nothing**.

Note, that after you've called the **EndShape**, to start building a new shape you have to initialize the current Basic shape of the group/page by calling the **BeginShape** method.

See Also BeginShape method

Equal Method

Equal Method

Copies all properties and contents of the source shape to the instance of a shape from the **Applies to** list.

Applies to: DPoint object**,** DRect object, Master object, ServObj object, Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**Equal** (*srcObject*)

The **Equal** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
srcObject	Required. An expression that returns an object of the same type as *object*. The source object for copying.
booleanRet	Optional. A **Boolean** type variable.

ExcelColorValue Method

ExcelColorValue Method

Returns an instance of **Color,** which contains information about the color, the value of which are located at the specified position in the table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *color* =] *object*. **ExcelColorValue** *(dsIndex, sheet, row, col)*

The **ExcelColorValue** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view XLS file specified data source object (shape).
col	Required. An expression that returns a **Long** value. The column number for the address in the table view XLS file specified data source object (shape).
ret	Optional. A **Color** type variable.

Remarks

The numbering of pages, lines, and stobtsov in the table view XLS file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. Color value in the data source must be specified in a Web format (# 00000000).

Example

Getting the color value that is in the first sheet in the second row in the third column, the second source of data sources in the collection of data object (shape).

```
dim res as Color
res = thisShape.ExcelColorValue (2,1,2,3)
if res.isRGB then
trace res.Red
trace res.Green
trace res.Blue
endif
```

DataSource

See Also

ExcelGetColumnForKey Method

ExcelGetColumnForKey Method

Returns the column number, found by searching on a key in a table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **ExcelGetColumnForKey** *(dsIndex, sheet, keyRow, keyStr)*

The **ExcelGetColumnForKey** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
keyRow	Required. An expression that returns a **Long** value. The line number with the key word for the address in the table view XLS file specified data source object (shape).
keyStr	Required. An expression that returns a **String** value. Keyword search.
ret	Optional. A **Long** type variable.

Remarks

The numbering of pages, lines, and stobtsov in the table view XLS file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. In case of addressing the range of the table, or in the absence of data, or if the keyword is not found, returns 0.

Example

Getting the column number on the first sheet in the third row of the second source of data in the collection of data source object (shape). Find the column is the keyword "black".

```
dim res as Long
res = thisShape.ExcelGetColumnForKey (2,1,3, "black")
trace res
```

DataSource

See Also

ExcelMinRowLength Method

ExcelMinRowLength Method

Returns the minimum number of lines (from all the rows) in the table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *length* =] *object.* **ExcelMinRowLength** *(dsIndex, sheet)*

The **ExcelMinRowLength** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
length	Optional. A **Long** type variable.

Remarks

The numbering of pages in the source XLS starts at 1. The numbering of the data sources in the collection of data sources, the object starts at 1.

Example

Obtaining the minimum number of line items (of all lines) from the first source of data from the first sheet in the collection of data sources of the object (shape).

```
dim num as Integer
num = thisShape.ExcelMinRowLength (1,1)
trace num
```

DataSource

See Also

ExcelRowLength Method

ExcelRowLength Method

Returns the number of elements in the specified row in a table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *num* =] *object*. **ExcelRowLength** *(dsIndex, sheet, row)*

The **ExcelRowLength** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view XLS file specified data source object (shape).
num	Optional. A **Long** type variable.

Remarks

The numbering of pages in the source XLS starts at 1. The numbering of the data sources in the collection of data sources, the object starts at 1.

Example

Getting the number of elements of the last line on the first page of the first data source in the collection of data sources, the object (shape).

```
dim num as Integer
num = thisShape.ExcelRowNum (1,1)
trace num
num = thisShape.ExcelRowLength (1,1, num)
trace num
```

DataSource

See Also

ExcelRowMaxElement Method

ExcelRowMaxElement Method

Returns the maximum element of the row in the table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **ExcelRowMaxElement** *(dsIndex, sheet, row, defVal)*

The **ExcelRowMaxElement** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view XLS file specified data source object (shape).
defVal	Required. An expression that returns a **Double** value. The default value.
ret	Optional. A **Double** type variable.

Remarks

The numbering of pages, lines, and stobtsov in the table view XLS file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the maximum value of the data, which are located on the first sheet in the third row of the second source of data in the collection of data source object (shape).

```
dim res as Double
res = thisShape.ExcelRowMaxElement (2,1,3, -1.2)
trace res
```

DataSource

See Also

ExcelRowMinElement Method

ExcelRowMinElement Method

Returns the minimum element of the row in the table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **ExcelRowMinElement** *(dsIndex, sheet, row, defVal)*

The **ExcelRowMinElement** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view XLS file specified data source object (shape).
defVal	Required. An expression that returns a **Double** value. The default value.
ret	Optional. A **Double** type variable.

Remarks

The numbering of pages, lines, and stobtsov in the table view XLS file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the minimum value of data that resides on the first sheet in the third row of the second source of data in the collection of data source object (shape).

```
dim res as Double
res = thisShape.ExcelRowMinElement (2,1,3, -1.8)
trace res
```

DataSource

See Also

ExcelRowNum Method

ExcelRowNum Method

Returns the number of rows in a table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax
[[Let] *num* =] *object.* **ExcelRowNum** *(dsIndex, sheet)*

The **ExcelRowNum** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
num	Optional. A **Long** type variable.

Remarks

ExcelRowNum method returns the number of rows in a table view XLS file data source object (shape). The numbering of pages in the source XLS starts at 1. The numbering of the data sources in the collection of data sources, the object starts at 1.

Example

Getting the number of rows in a table view XLS file on the first page of the first data source in the collection of data sources, the object (shape).

```
dim num as Integer
num = thisShape.ExcelRowNum (1,1)
trace num
```

See Also

DataSource

ExcelTextForKey Method

ExcelTextForKey Method

Returns the text found by searching on a key in a table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax
[[Let] *ret* =] *object.* **ExcelTextForKey** *(dsIndex, sheet, keyRow, keyStr, valueRow, defVal)*

The **ExcelTextForKey** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
keyRow	Required. An expression that returns a **Long** value. The line number with the key word for the address in the table view XLS file specified data source object (shape).
keyStr	Required. An expression that returns a **String** value. Keyword search.
valueRow	Required. An expression that returns a **Long** value. The line number of the desired value for the address in the table view XLS file specified data source object (shape).
defVal	Required. An expression that returns a **String** value. The default value.
ret	Optional. A **String** type variable.

Remarks

The numbering of pages, lines, and stobtsov in the table view XLS file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the data that resides on the first sheet in the third row of the second source of data in the collection of data source object (shape). Find the column is the keyword "black", which is located on the second line of the source.

```
dim res as String
res = thisShape.ExcelTextForKey (2,1,2, "black", 3, "Error")
trace res
```

See Also

DataSource

ExcelText Method

ExcelText Method

Returns the text written in a specified position in the table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **ExcelText** *(dsIndex, sheet, row, col, defVal)*

The **ExcelText** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view XLS file specified data source object (shape).
col	Required. An expression that returns a **Long** value. The column number for the address in the table view XLS file specified data source object (shape).
defVal	Required. An expression that returns a **String** value. The default value.
ret	Optional. A **String** type variable.

Remarks

The numbering of pages, lines, and stobtsov in the table view XLS file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the data that resides on the first sheet in the fourth row and first column of the third source of data in the collection of data source object (shape).

```
dim res as String
res = thisShape.ExcelText (3,1,4,1, "Error")
trace res
```

DataSource

See Also

ExcelValueDForKey Method

ExcelValueDForKey Method

Returns the value found using the search key in a table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **ExcelValueDForKey** *(dsIndex, sheet, keyRow, keyStr, valueRow, defVal)*

The **ExcelValueDForKey** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
keyRow	Required. An expression that returns a **Long** value. The line number with the key word for the address in the table view XLS file specified data source object (shape).
keyStr	Required. An expression that returns a **String** value. Keyword search.
valueRow	Required. An expression that returns a **Long** value. The line number of the desired value for the address in the table view XLS file specified data source object (shape).
defVal	Required. An expression that returns a **Double** value. The default value.
ret	Optional. A **Double** type variable.

Remarks

The numbering of pages, lines, and stobtsov in the table view XLS file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the data that resides on the first sheet in the first line of the third source of data in the collection of data source object (shape). Find the column is the keyword "green", which is located on the second line of the source.

```
dim res as Double
res = thisShape.ExcelValueDForKey (3,1,2, "green", 1, -1.8)
trace res
```

DataSource

See Also

ExcelValueD Method

Gets a value that is at the specified position in the table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **ExcelValueD** *(dsIndex, sheet, row, col, defVal)*

The **ExcelValueD** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view XLS file specified data source object (shape).
col	Required. An expression that returns a **Long** value. The column number for the address in the table view XLS file specified data source object (shape).
defVal	Required. An expression that returns a **Double** value. The default value.
ret	Optional. A **Double** type variable.

Remarks

The numbering of pages, lines, and stobtsov in the table view XLS file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the data that resides on the first sheet in the first row and fourth column of the third source of data in the collection of data source object (shape).

```
dim res as Double
res = thisShape.ExcelValueD (3,1,1,4, -1.5)
trace res
```

DataSource

See Also

ExcelValueForKey Method

ExcelValueForKey Method

Returns the integer value found by searching on a key in a table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **ExcelValueForKey** *(dsIndex, sheet, keyRow, keyStr, valueRow, defVal)*

The **ExcelValueForKey** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
keyRow	Required. An expression that returns a **Long** value. The line number with the key word for the address in the table view XLS file specified data source object (shape).
keyStr	Required. An expression that returns a **String** value. Keyword search.
valueRow	Required. An expression that returns a **Long** value. The line number of the desired value for the address in the table view XLS file specified data source object (shape).
defVal	Required. An expression that returns a **Long** value. The default value.
ret	Optional. A **Long** type variable.

Remarks

The numbering of pages, lines, and stobtsov in the table view XLS file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the data that resides on the first sheet in the third row of the second source of data in the collection of data source object (shape). Find the column is the keyword "black", which is located on the second line of the source.

```
dim res as Long
res = thisShape.ExcelValueForKey (2,1,2, "black", 3, -1)
trace res
```

See Also

DataSource

ExcelValueType Method

ExcelValueType Method

Returns the type of data that resides in the specified position in the table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **ExcelValueType** *(dsIndex, sheet, row, col)*

The **ExcelValueType** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view XLS file specified data source object (shape).
col	Required. An expression that returns a **Long** value. The column number for the address in the table view XLS file specified data source object (shape).
ret	Optional. A **Long** type variable.

Remarks

The numbering of pages, lines, and stobtsov in the table view XLS file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1.Interpretation of the return value: 0 - Void; 1 - String; 2 - Integer; 3 - Float; 4 - Color; 5 - Date;

Example

Getting the data type, which are located on the first sheet in the second row in the third column, the second source of data in the collection of data source object (shape).

```
dim res as Long
```

```
res = thisShape.ExcelValueType (2,1,2,3)
trace res
```

See Also [DataSource]

ExcelValue Method

ExcelValue Method

Returns an integer value that is at the specified position in the table view XLS file specified data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object*. **ExcelValue** *(dsIndex, sheet, row, col, defVal)*

The **ExcelValue** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
sheet	Required. An expression that returns a **Long** value. XLS file sheet number of the specified data source object (shape).
row	Required. An expression that returns a **Long** value. The line number for the address in the table view XLS file specified data source object (shape).
col	Required. An expression that returns a **Long** value. The column number for the address in the table view XLS file specified data source object (shape).
defVal	Required. An expression that returns a **Long** value. The default value.
ret	Optional. A **Long** type variable.

Remarks

The numbering of pages, lines, and stobtsov in the table view XLS file data source object (shape) starts with 1. The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the data that resides on the first sheet in the third row and fourth column in the third source of data in the collection of data source object (shape).

```
dim res as Long
res = thisShape.ExcelValue (3,1,3,4, -1)
trace res
```

See Also

DataSource

Export Method

Export Method

Exports the document to one of the file formats, supported by ConceptDraw.

Applies to: Document object

Syntax

[[Let] *booleanRet* =] *object.* **Export** *(fileName, formatType, [showSaveDlg], [showExportSetupDlg])*

The **Export** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Document** object.
fileName	Required. An expression that returns a **String** value. Represents the filename and path (full or relative) of the file, to which the document is being exported.
formatType	Required. An expression that returns a **Long** value. Specifies the format of the file, the which the document is being exported.
showSaveDlg	Optional. An expression that returns a **Boolean** value. A flag that specifies whether the file save dialog must be displayed. The default value is **False.**
showExportSetupDlg	Optional. An expression that returns a **Boolean** value. A flag that specifies whether to display the dialog with settings for the appropriate export format.The default value is **False.**
booleanRet	Optional. A **Boolean** type variable.

Remarks

If the file was exported successfully, the **Export** method returns **True,** otherwise it returns **False.**

The file is not exported if the provided filename *(fileName)* is not valid for the platform, on which ConceptDraw is running, or if the specified file format *(formatType)* is not supported by ConceptDraw. The list of supported file formats and corresponding ConceptDraw Basic Constants CAN be found here .

An inverse method to **Export** is the **Import** method, which imports a file of one of the formats, supported by ConceptDraw.

See Also Import / Export Constants , Import Method

FileText Method

FileText Method

Returns the text written in that text file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **FileText** *(dsIndex, startPos, count, defVal)*

The **FileText** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
startPos	Required. An expression that returns a **Long** value. The character position in a text file of the specified data source object (shape), which will begin reading.
count	Required. An expression that returns a **Long** value. Determines how many characters to read from a text file of the specified data source, starting from the position of the character defined startPos.
defVal	Required. An expression that returns a **String** value. The default value.
ret	Optional. A **String** type variable.

Remarks

The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set for the case of lack of data. If the count parameter is 0, then read out the entire text of the position startPos until the end of the file.

Example

Getting the data that resides in a text file, the third source of data in the collection of data source object (shape). Reading of data starts with the fourth character from the beginning of the file and read 10 characters.

```
dim res as String
res = thisShape.FileText (4,10, "Error")
trace res
```

See Also

DataSource object , CSVText , ExcelText , XPathText

FindFontByName method

FindFontByName Method

Searches for a font by its name in the font collection of the document. Returns the index of the found font in the font collection of the document.

Applies to: Document object

Syntax

[[**Let**] *index* =] *object*.**FindFontByName** (*fontName*)

The **FindFontByName** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Document** object.
fontName	Required. An expression that returns a **String** value. Represents the font name.
index	Optional. A **Long** type variable.

Remarks

If the font with the specified name is not found in the font collection of the document, the **FindFontByName** method returns **0**. An inverse operation to **FindFontByName** is the **FontName** method, which returns the name of the font by its index in the font collection of the document.

See Also FontName method, FontsNum method

FindLib Method (Application object)

FindLib Method (Application object)

Searches for a library among the libraries open in the application. Returns the index of the found library in the library collection of the application.

Applies to: Application object

Syntax

[[**Set**] *indexRet* =] *object*.**FindLib** (*inLib*)

The **FindLib** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
inLib	Required. An expression that returns a **Library** object.
indexRet	Optional. A **Long** type variable.

Remarks

If the specified library has not been found in the library collection of the application, the **FindLib** method returns **0**. The inverse method for **FindLib** is the **Lib** method, which returns a library by its index in the library collection of the application.

Example

This example contains an application-level script. The script closes the library if it's the second library in the library collection of the application.

```
Sub Close_if_2nd_Lib ( inLib As Library )
    If thisApp.FindLib( inLib ) = 2 Then
        thisApp.CloseLib( inLib )
    End If
End Sub
Close_if_2nd_Lib( thisApp.ActiveLibWnd.Library )
```

See Also	CloseLib method, CreateNewLib method, FindLib method, Lib method, LibByName method, LibsNum method, OpenLib method, Library object

FindLib Method (Window object)

FindLib Method (Window object)

Returns the index of the library in the library collection of the window.

Applies to: Window object

Syntax

[[**Let**] *indexRet* =] *object*.**FindLib** (*inLib*)

The **FindLib** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Window** object.
inLib	Required. An expression that returns a **Library** object.
indexRet	Optional. A **Long** type variable.

Remarks

This method is only effective if the library is a library window (see the **Type** property). For windows of other types the **FindLib** method always returns **0**.

If the specified library has not been found in the library collection of the window, the **FindLib** method returns **0**. The inverse method for **FindLib** is the **Lib** method, which returns a **Library** object by its index in the library collection of the window.

See Also	Type property, Lib method, LibByName method, LibsNum method, Library object

FindMaster Method

FindMaster Method

Searches for a master object in the master object collection of the library.

Applies to: Library object

Syntax

[[**Let**] *index* =] *object*.**FindMaster** (*masterObj*)

The **FindMaster** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Library** object.
masterObj	Required. An expression that returns a **Master** object. The library object to be found.
index	Optional. A **Long** type variable.

Remarks

If the search has been successful, the **FindMaster** method returns the index of the specified master object in the master object collection of the library. Otherwise the method returns **0**. An inverse method for this method is the **Master** method, which returns a master object by its index in the master object collection of the library.

See Also	AddMaster method, FindMaster method, Master method, MasterByName method, MastersNum method, RemoveMaster method, RemoveMasterByName method

FindMenuItem Method

FindMenuItem Method

This method searches for an instance of the **MenuItem** object in the menu item collection of the menu.

Applies to: Menu object

Syntax

[[**Let**] *indexRet* =] *object*.**FindMenuItem** (*menuItemObj*)

The **MenuItem** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Menu** object.
menuItemObj	Required. An expression, that returns an instance of the **MenuItem** object. Represents a menu item, which index is to be found.
indexRet	Optional. A **Long** type variable.

Remarks

If the search was successful, the **FindMenuItem** method returns the index of the specified menu itme in the menu item collection of the menu. Otherwise, it returns **0**.

See Also

FindPage Method

FindPage Method

Searches for a page in the page collection of the document. Returns the index of the page in the page collection of the document.

Applies to: Document object

Syntax

[[**Let**] *indexRet* =] *object*.**FindPage** (*pageObj*)

The **FindPage** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns a **Document** object.
pageObj	Required. An expression, that returns a **Page** object. Represents the page which index is to be returned.
indexRet	Optional. A **Long** type variable.

Remarks

If the specified page was not found in the page collection of the document, the **FindPage** method returns **0**. The inverse method to **FindPage** is the **Page** method, which returns a page by its index in the page collection of the document.

Example

This example contains a page-level script. It draws a rectangle in the upper left corner of the page, which contains the number of the page in the page collection of the document. The **FindPage** method is used to find the number of the page.

```
thisPage.DrawRect( 0,0,100,100 ).Text = thisDoc.FindPage( thisPage )
```

See Also	AddPage method, Page method, PageByID method, PagesNum method, RemovePage method, RemovePageByID method, ReorderPage method, ReorderPageByID method, Page object

FindStyle Method

FindStyle Method

Searches for a style in the style collection of the document. Returns the index of the specified style in the style collection of the document.

Applies to: Document object

Syntax

[[**Set**] *indexRet* =] *object*.**FindStyle** (*styleObj*)

The **FindStyle** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
styleObj	Required. An expression, that returns an instance of the **Style** object. Specified the style, which index is to be returned.
indexRet	Optional **Long**.

Remarks

If the specified style was not found in the style collection of the document, the **FindStyle** method returns **0**.

Example

This example contains a document-level script. It uses the **FindStyle** method to demonstrate that a new style is added to the end of the style collection of the document. Two new styles are added, and then their indices are displayed. The index of the last style added is equal to the number of styles in the document.

```
' Declare variables
Dim new_style1 As Style
Dim new_style2 As Style
Dim index1 As Long
Dim index2 As Long
' Add two new styles
Set new_style1 = thisDoc.AddStyle("New_Style_1")
Set new_style2 = thisDoc.AddStyle("New_Style_2")
TRACE new_style1
TRACE new_style1
' Get indices of the added styles
Let index1 = thisDoc.FindStyle( new_style1 )
Let index2 = thisDoc.FindStyle( new_style2 )
' Display the indices to make sure
' that the index of the style added first is less
' than the index of the second added style
TRACE "index1 = " & index1
TRACE "index2 = " & index2
```

See Also	AddStyle method, RemoveStyle method, RemoveStyleByName method, RenameStyle method, Style method, StyleByName method, StylesNum method, Style object

FirstDoc Method

FirstDoc Method

Returns an instance of the **Document** object corresponding to the first document in the document collection of the application.

Applies to: Application object

Syntax

[[**Set**] *docRet* =] *object*.**FirstDoc** ()

The **FirstDoc** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Application** object.
docRet	Optional. A **Document** type variable.

Remarks

If there are no open documents in the application, the **FirstDoc** method returns **Nothing**. To get the next documents in the document collection, use the **NextDoc** method.

Example

This example contains an application-level script. The script saves and closes all open documents, using the **FirstDoc** method to go through documents.

```
Dim curDoc As Document
Set curDoc = thisApp.FirstDoc()  ' Get the first document
While curDoc <> Nothing
    curDoc.Save()                ' Save document in the current folder
    thisApp.CloseDoc( curDoc )   ' Close document
    curDoc = thisApp.FirstDoc()  ' Get next document
Wend
```

See Also	CloseDoc method, CreateNewDoc method, Doc method, DocByName method, DocsNum method, NextDoc method, OpenDoc method, Document object

FirstLibWindow Method

FirstLibWindow Method

Returns an instance of the **Window** object that corresponds to the first library window in the window collection of the application.

Applies to: Application object

Syntax

[[**Set** *libWindowRet* =] *object*.**FirstLibWindow** ()

The **FirstLibWindow** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Application** object.
libWindowRet	Optional. A **Window** type variable.

Remarks

If there are no library windows in the application, the **FirstLibWindow** method returns **Nothing**. To get the next windows in the window collection, use the **NextLibWindow** method.

Example

This example contains an application-level script. The script displayes the titles (the **Title** property) of all libraries in the first library window of the application.

```
Dim libWnd As Window                        ' Declare variables
Set libWnd = thisApp.FirstLibWindow()       ' Get first window
If libWnd <> Null Then
    For i=1 To libWnd.LibsNum()             ' For each library in the window
        TRACE libWnd.Lib(i).Title           ' display its title
    Next i
Else
    TRACE "There are no library windows!" ' If there are no library windows
End If
```

See Also	LibWindowByID method, LibWindowsNum method, NextLibWindow method, Window object

FirstView Method

FirstView Method

Returns an instance of the **Window** object that corresponds to the first window in the window collection of the document.

Applies to: Document object

Syntax

[[**Set**] *windowRet* =] *object*.**FirstView** ()

The **FirstView** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns a **Document** object.
windowRet	Optional. A **Window** type variable.

Remarks

Note, that the window collection of the document can include windows of the following types (the **Type** property): document view, table view, Basic view. It's convenient to use the **FirstView** method together with the **NextView** method to go through all windows of the document.

See Also	Type property, NextView method, ViewByID method, ViewsNum method, UpdateAllViews method, Window object

FontName Method

FontName Method

Returns the name of the font by its index in the font collection of the document.

Applies to: Document object

Syntax

[[**Let**] *fontNameRet* =] *object*.**FontName** (*index*)

The **FindStyle** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Document** object.
index	Required. An expression that returns a **Long** value. Represents the index of the font in the font collection of the document.
fontNameRet	Optional. A **String** type variable.

Remarks

If *index* is less than **1** or greater than the number of fonts in the font collection of the document, the **FontName** method returns an empty string. Use the **FontsNum** method to find out the number of the fonts. The inverse method to **FontName** is the **FindFontByName** method which returns the index of the font in the font collection by the specified font name.

See Also FindFontByName method, FontsNum method

FontsNum Method

FontsNum Method

Returns the number of the fonts in the font collection of the document.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**FontsNum** ()

The **FontsNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
countRet	Optional. A **Long** type variable.

Remarks

The list of fonts is built when the document is created. The number of the fonts depends on how many fonts are installed on the operating system. The **FontsNum** always returns a value equal or greater than **1**, because there's always at least 1 font on the operating system.

See Also FontName method, FindFontByName method

GeometriesNum Method

GeometriesNum Method

Returns the number of geometries in the shape.

Applies to: Shape object

Syntax
[[**Let**] *countRet* =] *object*.**GeometriesNum** ()

The **GeometriesNum** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
countRet	Optional. A **Long** type variable.

Remarks

For all 1D and 2D shapes the **GeometriesNum** method always returns a value equal to or greater than **1**, as such shapes always contain at least one geometry. For shapes of other types the method always returns **0**, as they don't contain geometries.

See Also AddGeometry method, Geometry method, RemoveGeometry method, Geometry object

Geometry Method

Geometry Method

Returns a **Geometry** object that corresponds to a geometry with the specified index in the geometry collection of the shape.

Applies to: Shape object

Syntax
[[**Set**] *geometryRet* =] *object*.**Geometry** (*index*)

The **Geometry** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the geometry in the geometry collection of the shape.
geometryRet	Optional. A **Geometry** type variable.

Remarks

If *index* is less than **1** or greater than the number of geometries of the shape, the **Geometry** method returns **Nothing**. To find out the number of geometries of the shape, use the **GeometriesNum** method.

See Also AddGeometry method, GeometriesNum method, RemoveGeometry method, Geometry object

GetBlack Method

GetBlack Method

An Integer value. Gets the value of the black component of the color regardless of the color scheme of the object.

Applies to objects: Color

Syntax
[[**Let**] *ret* =] *object*.**GetBlack** (*Doc*)

The **GetBlack** method syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
Doc	A reference to an instance of the Document object.
ret	An **Integer** type variable (range 1 - 100).

Remarks

You can also use the Black propery to get the value of the black component of the object's color. However, that property is only effective if the color of *object* is in the CMYK format. For correct transformation of an indexed color to the RGB or CMYK format for the specified document, the *Doc* parameter is used in the **GetBlack** method.

Example

This example shows how to find out the value of the black component of a rectangle's fill color (the color was specified in the RGB format).

```
dim s as shape
' Create Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
s.FillColor.SetRGB(30,210,80)  ' Change fill color in RGB format
s.PropertyChanged(CDPT_FILLCOLOR)
trace s.FillColor.GetBlack(thisDoc)  ' Display the value of the black
component
```

See Also Color Object

GetBlue Method

GetBlue Method

An Integer value. Gets the value of the blue component of the color regardless of the color scheme of the object.

Applies to objects: Color

Syntax

[[**Let**] *ret* =] *object*.**GetBlue** (*Doc*)

The **GetBlue** method syntax has these Elements:

Element	Description
object	A reference to an instance of the object.
Doc	A reference to an instance of the Document object.
ret	An **Integer** type variable (range 0 - 255).

Remarks

You can also use the Blue propery to get the value of the blue component of the object's color. However, that property is only effective if the color of *object* is in the RGB format. For correct transformation of an indexed color to the RGB or CMYK format for the specified document, the *Doc* parameter is used in the **GetBlue** method.

Example

This example shows how to find out the value of the blue component of a rectangle's fill color (the color was specified in the CMYK format).

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
s.FillColor.SetCMYK(30,10,70,35)  ' Change fill color in CMYK format
s.PropertyChanged(CDPT_FILLCOLOR)
trace s.FillColor.GetBlue(thisDoc)  ' Get the value of the blue component
```

See Also Color Object

GetBooleanProperty Method

GetBooleanProperty Method

Returns the value of a **Boolean** type property.

Applies to: Shape object

Syntax

[[**Let**] *ret* =] *object*.**GetBooleanProperty**(*propTag* [, *num*[, *geom*]])

The **GetBooleanProperty** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the Shape object.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.

geom	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.
ret	Optional. A variable that gets the value returned by the method.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or by using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag, num, geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all valid property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

GetByteProperty Method

GetByteProperty Method

Returns the value of a **Byte** type property.

Applies to: Shape object

Syntax

[[**Let**] *ret* =] *object*.**GetByteProperty**(*propTag* [, *num*[, *geom*]])

The **GetByteProperty** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a Shape object.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the shape.
num	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.
ret	Optional. A variable that gets the value returned by the method.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or by using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all valid property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

GetCharacterIndex Method

GetCharacterIndex Method

Returns the index of a character block, which contains a character with the specified index in the text string of the shape.

Applies to: Shape object

Syntax

[[**Let**] *indexRet* =] *object*.**GetCharacterIndex** (*iSymbol*)

The **GetCharacterIndex** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
iSymbol	Required. An expression that returns a **Long** value. The character index in the text string of the shape.
indexRet	Optional. A **Long** type variable.

Remarks

If *index* is less than **1** or greater than the number of character in the text string of the shape, the **GetCharacterIndex** method returns **0**.

See Also	Character method, CharactersNum method, RemoveCharacter method, SetCharColor method, SetCharFont method, SetCharHyperlink method, SetCharLanguage method, SetCharPos method, SetCharSize method, SetCharSpacing method, SetCharStyle method, Character object

GetCyan Method

GetCyan Method

An Integer value. Gets the value of the cyan component of the color regardless of the color scheme of the object.

Applies to objects: Color

Syntax

[[**Let**] *ret* =] *object*.**GetCyan** (*Doc*)

The **GetCyan** method syntax has these Elements:

Element	Description

object	A reference to an instance of the object.
Doc	A reference to an instance of the Document object.
ret	An **Integer** type variable (range 1 - 100).

Remarks

You can also use the Cyan propery to get the value of the cyan component of the object's color. However, that property is only effective if the color of *object* is in the CMYK format. For correct transformation of an indexed color to the RGB or CMYK format for the specified document, the *Doc* parameter is used in the **GetCyan** method.

Example

This example shows how to find out the value of the cyan component of a rectangle's fill color (the color was specified in the RGB format).

```
dim s as shape
' Create Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
s.FillColor.SetRGB(30,210,80)  ' Change fill color in RGB format
s.PropertyChanged(CDPT_FILLCOLOR)
trace s.FillColor.GetCyan(thisDoc)  ' Display the value of the cyan component
```

See Also

Color Object

GetDoubleProperty Method

GetDoubleProperty Method

Returns the value of a **Double** type property.

Applies to objects: Shape, ServObj

Syntax

[[**Let**] *ret* =] *object*.**GetDoubleProperty**(*propTag* [, *num*[, *geom*]])

The **GetDoubleProperty** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object from the **Applies to** list.

propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.
ret	Optional. . An expression that returns a **Double** value. A variable that gets the value returned by the method.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or by using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object and **ServObj** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all valid property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

GetGreen Method

GetGreen Method

An Integer value. Gets the value of the green component of the color regardless of the color scheme of the object.

Applies to objects: Color

Syntax

[[**Let**] *ret* =] *object*.**GetGreen** (*Doc*)

The **GetGreen** method syntax has these Elements:

Element	Description
object	A reference to an instance of the object.
Doc	A reference to an instance of the Document object.
ret	An **Integer** type variable (range 0 - 255).

Remarks

You can also use the Green propery to get the value of the green component of the object's color. However, that property is only effective if the color of *object* is in the RGB format. For correct transformation of an indexed color to the RGB or CMYK format for the specified document, the *Doc* parameter is used in the **GetGreen** method.

Example

This example shows how to find out the value of the green component of a rectangle's fill color (the color was specified in the CMYK format).

```
dim s as shape
' Create Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
s.FillColor.SetCMYK(30,10,70,35)   ' Change fill color in CMYK format
s.PropertyChanged(CDPT_FILLCOLOR)
trace s.FillColor.GetGreen(thisDoc)   ' Display the value of the green
component
```

See Also Color Object

GetHeight Method

GetHeight Method

A Double value. Returns the height of the rectangle.

Applies to objects: DRect

Syntax

[[**Let**] *width* =] *object*.**GetHeight**()

The **GetHeight** method syntax has these Elements:

Element	Description
height	A Double type variable.
object	A reference to an instance of the object.

Example

This example is used to calculate the height of a rectangle, which coordinates are stored in MyObject.

```
Dim h as Double, MyObject as new DRect
' Set DRect object properties
MyObject.SetRect(30,30,100,90)
' Determine the height of  MyObject
w = MyObject.GetHeight()    ' h = 60
```

See Also

DRect Object, GetWidth Method

GetHyperlinkID Method

GetHyperlinkID Method

Returns the ID of a hyperlink, provided the hyperlink is present in the hyperlink collection of the document.

Applies to: Document object

Syntax

[[**Let**] *hyperlinkIDRet* =] *object*.**GetHyperlinkID** (*hyperlinkObj*)

The **GetHyperlinkID** method syntax has these Elements:

Element	Description

object	Required. A reference to an instance of the Document object.
hyperlinkObj	Required. An expression that returns a Hyperlink type value. Represents the hyperlink, which ID is to be returned.
hyperlinkIDRet	Optional. A Long type variable.

Remarks

If the specified hyperlink doesn't exist in the document, the method returns **0**.

See Also	AddHyperlinkToDocument Method, AddHyperlinkToFile Method, AddHyperlinkToPageShape Method, AddHyperlinkToURL Method, Hyperlink Method, HyperlinkByID Method, HyperlinksNum Method, RemoveUnusedHyperlinks Method, Hyperlink Object, Document Object

GetIndex Method

GetIndex Method

Returns the index of the object (shape) in a collection of objects (shapes) of the parent group.

Applies to: Shape object

Syntax
[[Let] *index* =] *object*. **GetIndex** ()

The **GetIndex** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
index	Optional. A **Long** type variable.

Remarks

If the *object* is an object directly on the document page, then the parent of this object is a page (Property **Page).** If an object is placed in a group, then its parent is a group of objects. The numbering of objects starting with 0. In case of error the method returns -1.

Example
```
dim index as Integer
```

```
index = thisShape.GetIndex()
trace index
```

See Also Page property, Parent property, SendFront method, SendBack method, StepBack method, StepFront method

GetIntegerProperty Method

GetIntegerProperty Method

Returns the value of a **Integer** type property.

Applies to objects: Shape

Syntax

[[**Let**] *ret* =] *object*.**GetIntegerProperty**(*propTag* [, *num*[, *geom*]])

The **GetIntegerProperty** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the Shape object.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.
ret	Optional. A variable that gets the value returned by the method.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or by using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property:

propTag, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all valid property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

GetLongProperty Method

GetLongProperty Method

Returns the value of a **Long** type property.

Applies to objects: Shape

Syntax

[[**Let**] *ret* =] *object*.**GetLongProperty**(*propTag* [, *num*[, *geom*]])

The **GetLongProperty** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the Shape object.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.
ret	Optional. A variable that gets the value returned by the method.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or by using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all valid property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

GetMagenta Method

GetMagenta Method

An Integer value. Gets the value of the magenta component of the color regardless of the color scheme of the object.

Applies to objects: Color

Syntax

[[**Let**] *ret* =] *object*.**GetMagenta** (*Doc*)

The **GetMagenta** method syntax has these Elements:

Element	Description
object	A reference to an instance of the object.
Doc	A reference to an instance of the Document object.
ret	An **Integer** type variable (range 1 - 100).

Remarks

You can also use the Magenta propery to get the value of the magenta component of the object's color. However, that property is only effective if the color of *object* is in the CMYK format. For correct transformation of an indexed color to the RGB or CMYK format for the specified document, the *Doc* parameter is used in the **GetMagenta** method.

Example

This example shows how to find out the value of the magenta component of a rectangle's fill color (the color was specified in the RGB format).

```
dim s as shape
' Create Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
s.FillColor.SetRGB(30,210,80)  ' Change fill color in RGB format
s.PropertyChanged(CDPT_FILLCOLOR)
trace s.FillColor.GetMagenta(thisDoc)  ' Display the value of the magenta
component
```

See Also Color Object

GetParagraphIndex Method

GetParagraphIndex Method

Returns the paragraph index in the paragraph collection of the shape by the specified character index in the text string of the shape.

Applies to: Shape object

Syntax

[[**Let**] *paragraphIndexRet* =] *object*.**GetParagraphIndex** (*symbolIndex*)

The **GetParagraphIndex** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
symbolIndex	Required. An expression that returns a **Long** value. The index of the character in the text string of the shape.
paragraphIndexRet	Optional. A **Long** type variable.

Remarks

If *index* is less than **1** or greater than the number of characters in the text string of the shape, the **GetParagraphIndex** method returns **0**.

See Also	Paragraph method, ParagraphsNum method, RemoveParagraph method, SetParaAfterSpacing method, SetParaBeforeSpacing method, SetParaFirstInd method, SetParaHAlign method, SetParaLeftInd method, SetParaLineSpacing method, SetParaRightInd method, Paragraph object

GetPropertyFormula Method

GetPropertyFormula Method

Returns the formula of the shape's property in the form of a string.

Applies to: Shape object, ServObj

Syntax

[[**Let**] *ret* =] *object*.**GetPropertyFormula**(*propTag* [, *num*[, *geom*]])

The **GetPropertyFormula** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object from the **Applies to** list.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.
ret	Optional. A variable that gets the string returned by the method.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object and **ServObj** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

GetRed Method

GetRed Method

An Integer value. Gets the value of the red component of the color regardless of the color scheme of the object.

Applies to objects: Color

Syntax

[[**Let**] *ret* =] *object*.**GetRed** (*Doc*)

The **GetRed** method syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
Doc	A reference to an instance of the Document object.
ret	An **Integer** type variable (range 0 - 255).

Remarks

You can also use the Red propery to get the value of the red component of the object's color. However, that property is only effective if the color of *object* is in the RGB format. For correct transformation of an indexed color to the RGB or CMYK format for the specified document, the *Doc* parameter is used in the **GetRed** method.

Example

This example shows how to find out the value of the red component of a rectangle's fill color (the color was specified in the CMYK format).

```
dim s as shape
' Create Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
s.FillColor.SetCMYK(30,10,70,35)   ' Change fill color in CMYK format
s.PropertyChanged(CDPT_FILLCOLOR)
trace s.FillColor.GetRed(thisDoc)   ' Display the value of the red component
```

See Also Color Object

GetSelectService Method

GetSelectService Method

Returns a service object from the collection-selected objects (shapes) is displayed in a window or group of pages to index.

Applies to: Window object

Syntax

[[Let] *seriseRet* =] *object.* **GetSelectService** *(index)*

The **GetSelectService** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Window** object.
index	Required. An expression that returns a **Long** value. Specifies the index-selected objects in the collection is displayed in a window or group.
seriseRet	Optional. A **ServObj** type variable.

Remarks

This method is only effective if the window is of the document view type (see the **Type** property). For windows of all other types, the **GetSelectService** method always returns **NULL.**

If the specified index (parameter index) is correct for-selected collection of objects (shapes) is displayed in a window or group of pages, the **GetSelectService** method vozvraschet service object with the index.Otherwise, the method returns NULL.

See Also	ServObj Object, ID Property, Property Type, Deselect Method , Method DeselectAll , GetSelectedShape Method , Method Select, SelectAll Method, Method SelectedNum

GetSelectShape Method

GetSelectShape Method

Returns an instance of the **Shape** object, that represents a shape, associated with an instance of an object from the **Applies to** list.

Applies to: Window object

Syntax

[[Let] *shapeRet* =] *object.* **GetSelectShape** *(shapeIndex)*

The **GetSelectShape** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Window** object.
shapeIndex	Required. An expression that returns a **Long** value. Specifies the index of the shape-selected objects in the collection is displayed in a window or group.
shapeRet	Optional. A **Shape** type variable.

Remarks

This method is only effective if the window is of the document view type (see the **Type** property). For windows of all other types, the **GetSelectShape** method always returns **NULL.**

If the specified index (parameter shapeIndex) is valid for a collection-selected objects (shapes) is displayed in a window or group of pages, the **GetSelectShape** method returns an object with that index.Otherwise, the method returns NULL.

Example

The example demonstrates how you can get past the account-selected object from the collection-selected objects (shapes) is displayed in a window or group.

```
dim sh as Shape
dim num as Long
num = Thisdoc.ActiveView.SelectedNum ()
if num> 0 then
sh = Thisdoc.ActiveView.GetSelectedShape (num)
trace sh.ID
else
trace "No selected"
end if
```

See Also

Shape object, ID Property, Property Type, Deselect Method , Method DeselectAll ,GetSelectedService Method , Method Select, SelectAll Method, Method SelectedNum

GetShapeByName Method

GetShapeByName Method

Searches for a shape with the specified name **(Name** property) in the shape collection of the group / page. Returns the position of the found object (shape) in a collection of objects (shapes).

Applies to: Page object, Shape object

Syntax

[[Set] *pos* =] *object.* **GetShapeByName** *(name, start, end)*

The **GetShapeByName** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object from the **Applies to** list.
name	Required. An expression that returns a **String** value. The object name to search.
start	Required. An expression that returns a **Integer** value. Start position in the collection of objects (shapes).
end	Optional. An expression that returns a **Integer** value. The final position in the collection of objects (shapes).
pos	Optional. A **Long** type variable.

Remarks

If an object (shape) with the specified name is not found in the collection of objects (shapes), or the final position of the object passed by value over the starting, the method returns 0 **GetShapeByName.** If the end position is not specified, or equal to -1, the search facility will be made from the starting position until the end of a collection of objects.

Example

This example contains a shape-level script. In the first example of an object (shape), which has the name "FirstFindingObject", is searched, starting at position 1 and finishing fifth position in the collection of objects on the page. In the second example of an object (shape), which has the name "SecondFindingObject", sought, starting from the zero position until the end of the collection in a group of objects.

```
dim pos as Long
pos = thisPage.GetShapeByName ("FirstFindingObject", 1,5)
or
pos = thisShape.GetShapeByName ("SecondFindingObject", 0)
```

See Also	ShapeByID method, RemoveAllShapes method, RemoveShape method,RemoveShapeByID method, ReorderShape method, ReorderShapeByID method, Shape method, ShapesNum method

GetSingleProperty Method

GetSingleProperty Method

Returns the value of a **Single** type property.

Applies to objects: Shape

Syntax

[[Let] *ret* **=]** *object.* **GetSingleProperty** *(propTag* [, *num* [, *geom]])*

The **GetSingleProperty** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the Shape object.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. An expression that returns a **Long** value. An additional identifying argument.It's used for specifying properties from collections of the object.
geom	Optional. An expression that returns a **Long** value. An additional identifying argument.It's used for specifying properties from geometry collections of the object.
ret	Optional. A variable that gets the value returned by the method.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or by using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic Has a set of Constants That define all valid Property tags .

See Also GetByteProperty Method , Method GetBooleanProperty , GetIntegerProperty Method , Method GetLongProperty , GetSingleProperty Method , Method GetDoubleProperty , GetStringProperty Method , Method ColorProperty , SetByteProperty Method, Method SetBooleanProperty, SetIntegerProperty Method, Method SetLongProperty, SetSingleProperty Method, Method SetDoubleProperty, SetStringProperty Method, Method IsDefaultFormula, IsNullFormula Method, Method GetPropertyFormula , SetPropertyFormula Method, MethodSetDefaultFormula, SetNullFormula Method, Method RecalcProperty,PropertyChanged Method

GetStringProperty Method

GetStringProperty Method

Returns the value of a **String** type property.

Applies to objects: Shape, ServObj

Syntax

[[Let] *ret* **=]** *object.* **GetStringProperty** *(propTag* [, *num* [, *geom]])*

The **GetStringProperty** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. It is used only for the object Shape. An expression that returns a **Long** value.An additional identifying argument. It's used for specifying properties from collections of the object.

geom	Optional. It is used only for the object Shape. An expression that returns a **Long** value.An additional identifying argument. It's used for specifying properties from geometry collections of the object.
ret	Optional. A variable that gets the value returned by the method.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or by using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object and **ServObj** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic Has a set of Constants That define all valid Property tags .

See Also

GetByteProperty Method , Method GetBooleanProperty , GetIntegerProperty Method , Method GetLongProperty , GetSingleProperty Method , Method GetDoubleProperty , GetStringProperty Method , Method ColorProperty , SetByteProperty Method, Method SetBooleanProperty, SetIntegerProperty Method,Method SetLongProperty, SetSingleProperty Method, Method SetDoubleProperty,SetStringProperty Method, Method IsDefaultFormula, IsNullFormula Method,Method GetPropertyFormula , SetPropertyFormula Method, MethodSetDefaultFormula, SetNullFormula Method, Method RecalcProperty,PropertyChanged Method

GetWidth Method

GetWidth Method

A Double value. Returns the width of the rectangle.

Applies to objects: DRect

Syntax

[[Let] *width* =] *object*. **GetWidth** ()

The **GetWidth** method syntax has these Elements:

Element	**Description**
width	A Double type variable.
object	A reference to an instance of the object.

Example

This example is used to calculate the width of a rectangle, which coordinates are stored in MyObject.

```
Dim w as Double, MyObject as new DRect
'Set DRect object properties
MyObject.SetRect (30,30,100,90)
'Determine the width of MyObject
w = MyObject.GetWidth () 'w = 70
```

See Also DRect Object , Method GetHeight

GetYellow Method

GetYellow Method

An Integer value. Gets the value of the yellow component of the color regardless of the color scheme of the object.

Applies to objects: Color

Syntax

[[Let] *ret* =] *object*. **GetYellow** *(Doc)*

The **GetYellow** method syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
Doc	A Reference to an instance of the Document object.
ret	An **Integer** type variable (range 1 - 100).

Remarks

You can also use the Yellow propery to get the value of the yellow component of the object's color. However, that property is only effective if the color of *objest* is in the CMYK format. For correct transformation of an indexed color to the RGB or CMYK format for the specified document, the *Doc* parameter is used in the **GetYellow** method.

Example

This example shows how to find out the value of the yellow component of a rectangle's fill color (the color was specified in the RGB format).

```
dim s as shape
'Create Shape object
```

```
s = thisDoc.ActivePage.DrawRect (100,100,1000,1000)
s.FillColor.SetRGB (30,210,80) 'Change fill color in RGB format
s.PropertyChanged (CDPT_FILLCOLOR)
trace s.FillColor.GetYellow (thisDoc) 'Display the value of the yellow
component
```

See Also Color Object

GPtoLP Method

GPtoLP Method

Performs the conversion of the coordinates of the coordinate system of the parent object (shape) (group or page) in the local coordinate system of (this) object (shape).

Applies to: Shape object

Syntax

object.**GPtoLP** (*srcPoint*)

The **GPtoLP** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
srcPoint	Required. A **DPoint** type variable. The coordinates of the point.

Remarks

The global coordinate system with respect to the coordinate system of the given shape is the local coordinate system of the parent group in the case when the shape is inside a group. If the shape is not Element of a group, the global coordinate system coincides with the global coordinate system.

This method modifies the input argument *srcPoint* and uses it to return the resulting coordinates. The coordinates are measured in internal units (**InternalUnit**).

See Also LAtoWA method, LPtoGP method, LPtoWP method, WPtoLP method

HyperlinkByID Method

Searches for a hyperlink by the specified ID (the **ID** property) in the hyperlink collection of the document. Returns an instance of the **Hyperlink** object, corresponding to the found hyperlink.

Applies to: Document object

Syntax

[[**Set**] *hyperlinkRet* =] *object*.**HyperlinkByID** (*hyperlinkID*)

The **HyperlinkByID** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Document** object.
hyperlinkID	Required. An expression that returns a **Long** value. Represents the ID of the hyperlink being searched.
hyperlinkRet	Optional. A **Hyperlink** type variable.

Remarks

If the hyperlink with the specified *hyperlinkID* wasn't found in the document, the method returns **Nothing**.

Example

This example contains a page-level script. It displays the list of all hyperlinks which exist on the page (**thisPage**). The **HyperlinkByID** method is used to search for the hyperlink by the hyperlink ID taken from the shape (see the Hyperlink property).

```
' Declare variables
Dim linkID As Long
Dim hlink As Hyperlink
' Loops though all shapes on the page (thisPage)
For i=1 To thisPage.ShapesNum()
    ' Get ID of the shape's hyperlink
    linkID = thisPage.Shape(i).Hyperlink
    ' Search for hyperlink with specified ID in the hyperlink
    ' collection of the document
    Set hlink = thisDoc.HyperlinkByID( linkID )
    If hlink <> Null Then
        ' If hyperlink found, display its properties
        TRACE "Shape_" & i & "  " & hlink
        TRACE "   ID        = " & hlink.ID
        TRACE "   LinkType  = " & hlink.LinkType
        TRACE "   Address   = " & hlink.Address
        TRACE "   LocalPath = " & hlink.LocalPath
        TRACE "   PageID    = " & hlink.PageID
        TRACE "   ShapeID   = " & hlink.ShapeID
    End If
Next i
```

See Also ID property, Hyperlink property, AddHyperlinkToDocument method, AddHyperlinkToFile method, AddHyperlinkToPageShape method, AddHyperlinkToURL method, Hyperlink method, HyperlinksNum method, RemoveUnusedHyperlinks method, Hyperlink object

HyperlinksNum Method

HyperlinksNum Method

Returns the number of hyperlinks in the hyperlink collection of the document.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**HyperlinksNum** ()

The **HyperlinksNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
countRet	Optional. A **Long** type variable.

Remarks

Note, that the number of hyperlinks is increased when new hyperlinks are added to the collection and is decreased when unused hyperlinks are deleted (see the RemoveUnusedHyperlinks method). If there are no hyperlinks in the document, the **HyperlinksNum** method returns **0**.

Example

This example contains a document-level script. The script calculates the number of hyperlinks in the hyperlink collection of the document for each of the three hyperlink types: **cdLinkToFile**, **cdLinkToURL**, **cdLinkToPageShape**. The results of the calculation and the total number of hyperlinks in the document are displayed on the screen.

```
' Declare variables
Dim n_LinkToFile As Integer
Dim n_LinkToUrl  As Integer
Dim n_LinkToPageShape As Integer
' Initialize counters with zeros
n_LinkToFile = 0
n_LinkToUrl  = 0
n_LinkToPageShape = 0
' Loop through all hyperlinks in the hyperlink
' collection of the document
For i=1 To thisDoc.HyperlinksNum()
    ' Calculate the amount of hyperlinks of each type
    Select Case thisDoc.Hyperlink(i).LinkType
    Case cdLinkToFile
        n_LinkToFile = n_LinkToFile + 1
    Case cdLinkToURL
        n_LinkToUrl = n_LinkToUrl + 1
    Case cdLinkToPageShape
        n_LinkToPageShape = n_LinkToPageShape + 1
    End Select
Next i
' Display the results
TRACE "Number of:"
TRACE "Links to file = " & n_LinkToFile
TRACE "Links to URL = " & n_LinkToUrl
TRACE "Links to Page Or Shape = " & n_LinkToPageShape
TRACE "Total number of links = " & thisDoc.HyperlinksNum()
```

See Also	AddHyperlinkToDocument metohd, AddHyperlinkToFile metohd, AddHyperlinkToPageShape metohd, AddHyperlinkToURL method, Hyperlink method, HyperlinkByID method, RemoveUnusedHyperlinks method, Hyperlink object

Hyperlink Method

Hyperlink Method

Returns an instance of the **Hyperlink** object by the index of the hyperlink in the hyperlink collection of the document.

Applies to: Document object

Syntax

[[**Set**] *hyperlinkRet* =] *object*.**Hyperlink** (*index*)

The **Hyperlink** method syntax has these Elements:

Element	Description

object	Required. An expression, that returns an instance of the **Document** object.
index	Required. An expression that returns a **Long** value. Represents the index of the hyperlink in the hyperlink collection of the document.
hyperlinkRet	Optional. A **Hyperlink** type variable.

Remarks

If *index* is less than **1** or greater than the number of hyperlinks in the hyperlink collection of the document, the **Hyperlink** method returns **Nothing**. Use the **HyperlinksNum** method to find out the number of hyperlinks in the hyperlink collection of the document.

Example

This example contains a document-level script. It displays the list of properties of each hyperlink in the current document.

```
' Declare variables
Dim hlink As Hyperlink
' Loop through all hyperlinks
For i=1 To thisDoc.HyperlinksNum()
    ' Get next hyperlink from
    ' the hyperlink collection of the document
    Set hlink = thisDoc.Hyperlink(i)
    ' Display the hyperlink properties
    TRACE "Hyperlink_" & i & "  " & hlink
    TRACE "   ID        = " & hlink.ID
    TRACE "   LinkType  = " & hlink.LinkType
    TRACE "   Address   = " & hlink.Address
    TRACE "   LocalPath = " & hlink.LocalPath
    TRACE "   PageID    = " & hlink.PageID
    TRACE "   ShapeID   = " & hlink.ShapeID
Next i
```

See Also	AddHyperlinkToDocument method, AddHyperlinkToFile method, AddHyperlinkToPageShape method, AddHyperlinkToURL method, HyperlinkByID method, HyperlinksNum method, RemoveUnusedHyperlinks method, Hyperlink object

Import Method

Import Method

Imports a file of one of the formats, supported by ConceptDraw. Returns an instance of the **Document** object corresponding to the imported file.

Applies to: Application object

Syntax

[[**Set**] *docRet* =] *object*.**Import** (*fileName*, *formatType*, [*showSaveDlg*], [*showSettingsDlg*])

The **Import** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Application** object.
fileName	Required. An expression that returns a **String** value. Represents the name of the imported file.
formatType	Required. An expression that returns a **Long** value. Indicates the format of the imported file. This parameter must be equal to one of the pre-defined constants, which correspond the the file formats supported by ConceptDraw. If the format is indicated incorrectly, it is recognized automatically for raster files. For vector files format is not recognized.
showSaveDlg	Optional. An expression that returns a **Boolean** value. A flag that specifies whether to display the file open dialog. The default value is **False**.
showSettingsDlg	Optional. An expression that returns a **Boolean** value. A flag that specifies whether to display the import settings dialog for some file formats. This parameter is needed for the file formats, which can not be imported without information about their contents (for instance, which delimiter is used, etc). For example, such formats are **Outline** and **Flowdata**. When this flag is **False** the import settings are taken from the application settings which you can view and modify by choosing "Edit > Preferences". The defaut value is **False**.
docRet	Optional. A **Document** type variable.

Remarks

If the format of the imported file is specified as cdf_UNKNOWN (unknown file format) or set incorrectly, ConceptDraw tries to recognized the format automatically (except for the Outline and FlowData formats). If the file was imported successfully, tje **Import** method returns a reference to the instance of the **Document** object, which corresponds to the imported file. If the specified file couldn't be opened, the **Import** method returns **Nothing**.

The list of formats, supported in ConceptDraw and corresponding ConceptDraw Basic constants can be seen here.

The inverse method to **Import** is the **Export** method which saves a ConceptDraw document in a file with the specified format.

Example

This example contains application-level script.

```
thisApp.Import( "c:\ffffff.bmp", cdf_BMP, TRUE, TRUE )
```

See Also Import/Export constants, Export method, Document object

InflateRect Method

InflateRect Method

"Enlarges" the rectangle by the X and Y axes, calculates the coordinates of the object.

Applies to objects: DRect

Syntax

object.**InflateRect** (*x*, *y*)

The **InflateRect** method syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
x	A Double value, represents the offset for the left and right sides of the rectangle.
y	A Double value, represents the offset for the top and bottom sides of the rectangle.

Remarks

"Enlarging" the rectangle doesn't reposition its center. The following formulas are used to calculate the coordinates:

left = left - x; top = top - x; right = right + x; bottom = bottom + x

Example

```
' Create an instance of the object
Dim MyObject as new DRect
' Set left,top,right,bottom properties
MyObject.SetRect(200,200,1000,1000)
```

```
' Inflate rectangle
' After the operation the properties will be equal to:
' left - 100, top - 100, right - 1100, bottom - 1100
MyObject.InflateRect(100,100)
```

See Also DRect Object, DeflateRect Method

InsertPicture Method

InsertPicture Method

Creates a shape that contains the picture from the specified file and places it onto the page / into the group at the specified position. Returns a **Shape** object that corresponds to the created shape.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**InsertPicture** (*fileName*, *xInsert*, *yInsert*)

The **InsertPicture** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
fileName	Required. An expression that returns a **String** value. Full or relative path to the file that contains the image in a format that ConceptDraw can import.
xInsert	Required. An expression that returns a **Double** value. The X-coordinate of the rotation center (the **GPinX** property) for the created shape.
yInsert	Required. An expression that returns a **Double** value. The Y-coordinate of the rotation center (the **GPinY** property) for the created shape.
shapeRet	Optional. A **Shape** type variable.

Remarks

If the file with the specified *fileName* hasn't been found or couldn't be opened, the **InsertPicture** method doesn't create a new shape and returns **Nothing**.

See Also GPinX property, GPinY property, Import/Export constants

IntersectRect Method

IntersectRect Method

Calculates the coordinates of the rectangle, corresponding to the area of intersection of the two specified rectangles. Returns a Boolean value: TRUE if the rectangles intersect, otherwise FALSE.

Applies to objects: DRect

Syntax

object.**IntersectRect** (*inRect1*, *inRect2*)

The **IntersectRect** method syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
inRect1, *inRect2*	References to DRect objects.
res	A Boolean type variable.

Remarks

Note, that if the method returns FALSE, *object* properties are reset to zero.

Example

```
Dim outRect as new DRect, inRect1 as new DRect, inRect2 as new DRect, res as
Boolean
inRect1.SetRect(100,100,300,300)
inRect2.SetRect(200,200,400,400)
' intersect rect
' After the method has been called,
' outRect properties will become equal to 200,200,300,300
res = outRect.IntersectRect(inRect1,inRect2)  ' return TRUE
```

See Also DRect Object

IsDefaultFormula Method

IsDefaultFormula Method

Returns a **Boolean** type value. If the property has a formula, and it's marked as default, this method returns **True**, otherwise it returns **False**.

Applies to: Shape object, ServObj

Syntax

[[**Let**] *ret* =] *object*.**IsDefaultFormula**(*propTag* [, *num*[, *geom*]])

The **IsDefaultFormula** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.
ret	Optional. A variable that gets the value returned by the method.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object and **ServObj** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

IsEmpty Method

IsEmpty Method

Returns a Boolean value: TRUE, if the square of the rectangle is zero, otherwise FALSE.

Applies to objects: DRect

Syntax

[[**Let**] *res* =] *object*.**IsEmpty** ()

The **IsEmpty** method syntax has these Elements:

Element	Description
res	A Boolean type variable.
object	A reference to an instance of the DRect object.

Remarks

Note, that if the right / bottom values in the DRect object are less than the left / top values, the square of the rectangle is non-zero.

Example

This example demonstrates using the **IsEmpty** method.

```
Dim MyObject as new DRect, res as Boolean
```

```
'set DRect object properties
MyObject.SetRect(30,100,30,500)
res = MyObject.IsEmpty()   ' returns TRUE
```

See Also DRect Object

IsNullFormula Method

IsNullFormula Method

Returns a **Boolean** type value. If the property has no formula, the method returns **True**, otherwise it returns **False**.

Applies to objects: Shape, ServObj

Syntax

[[**Let**] *ret* =] *object*.**IsNullFormula**(*propTag* [, *num*[, *geom*]])

The **IsNullFormula** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.
ret	Optional. A variable that gets the value returned by the method.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or by using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object and **ServObj** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all valid property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

LAtoWA Method

LAtoWA Method

Converts the angle from the local coordinate system of the shape into the global coordinate system.

Applies to: Shape object

Syntax

[[**Let**] *retAngle* =]*object*.**LAtoWA** (*localAngle*)

The **WPtoLP** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
localAngle	Required. An expression that returns a **Double** value. The angle in the local coordinate system.
retAngle	Optional. A **Double** type variable. The resulting angle in the global coordinate system.

Remarks

The angle values are measured in **radians**.

See Also GPtoLp, LPtoGP method, LPtoWP method, WPtoLP method

LayerByID Method

LayerByID Method

Searches for a layer with the specified ID (ID property) in the layer collection of the document. Returns an instance of the **Layer** object that corresponds to the found layer.

Applies to: Document object

Syntax

[[**Set**] *layerRet* =] *object*.**LayerByID** (*layerID*)

The **LayerByID** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
layerID	Required. An expression that returns a **Long** value. Represents the ID (**ID** property) of the layer being searched.
layerRet	Optional. A **Layer** type variable.

Remarks

If there is no layer with the specified ID (**ID** property) in the layer collection of the document, the **LayerByID** method returns **Nothing**.

Example

This example contains a document-level script. The **LayerByID** method uses the layer ID taken from a shape on that layer.

```
' Declare variables
Dim shp1 As Shape
Dim shp2 As Shape
Dim layerID As Long
Dim player As Layer
```

```
' Set the first layer from the layer collection as active layer
thisDoc.ActiveLayer = thisDoc.Layer(1).ID
' Draw a rectangle on the active layer
Set shp1 = thisDoc.ActivePage.DrawRect(100,100,600,500)
shp1.Text = "Layer # 1"
' Use the LayerByID method to get the layer on which the shp1 shape
' was created
Set player = thisDoc.LayerByID( shp1.Layer )
' Set blue color for all shapes on the layer
player.Colored = True
player.Color.SetRGB( 0,0,255 )
' Make the second layer from the layer collection
'  active layer
thisDoc.ActiveLayer = thisDoc.Layer(2).ID
' Draw a rectangle on the active layer
Set shp2 = thisDoc.ActivePage.DrawRect(700,100,1200,500)
shp2.Text = "Layer # 2"
' Use the LayerByID method to get the layer on which the shp2 shape
' was created
Set player = thisDoc.LayerByID( shp2.Layer )
' ' Set red color for all shapes on the layer
player.Colored = True
player.Color.SetRGB( 255,0,0 )
```

See Also	ID property, AddLayer method, Layer method, LayerByName method, LayersNum method, RemoveLayer method, RemoveLayerByID method, Layer object

LayerByName Method

LayerByName Method

Searches for a layer with the specified name (the Name property) in the layer collection of the document. Returns an instance of the **Layer** object that corresponds to the first layer with the specified name, found in the layer collection.

Applies to: Document object

Syntax

[[**Set**] *layerRet* =] *object*.**LayerByName** (*layerName*)

The **LayerByID** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Document** object.

layerName	Required. An expression that returns a **String** value. Represents the name (the **Name** property) of the layer being searched.
layerRet	Optional. A **Layer** type variable.

Remarks

If there is no layer with the specified name in the layer collection of the document, the **LayerByName** method returns **Nothing**.

Example

This example contains a document-level script. The script shows how the **LayerByName** method is used to find the layer that was created earlier (with less index) among two layers with the same names.

```
' Declare variables
Dim layer1 As Layer
Dim layer2 As Layer
Dim reslayer As Layer
' Add two new layers to the document
Set layer1 = thisDoc.AddLayer()
Set layer2 = thisDoc.AddLayer()
' Give the same name to both layers
layer1.Name = "Layer Name"
layer2.Name = "Layer Name"
' Display the names to make sure they are the same.
TRACE layer1.Name
TRACE layer2.Name
' Search for layer with specified name
Set reslayer = thisDoc.LayerByName( "Layer Name" )
' Display the references to the instances of the Layer object,
' to make sure that the LayerByName method
' returned the reference to the first added layer -  layer1
TRACE "layer1 = " & layer1
TRACE "layer2 = " & layer2
TRACE "reslayer = " & reslayer
' Delete layers
thisDoc.RemoveLayerByID( layer1.ID )
thisDoc.RemoveLayerByID( layer2.ID )
```

See Also	AddLayer method, Layer method, LayerByID method, LayersNum method, RemoveLayer method, RemoveLayerByID method, Layer object

LayersNum Method

LayersNum Method

Returns the number of the layers in the layer collection of the document.

Applies to: Document object

Syntax

[[**Set**] *countRet* =] *object*.**LayersNum** ()

The **LayersNum** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Document** object.
countRet	Optional. A **Long** type variable.

Remarks

Never returns **0**, as there's always at least one layer in the document.

Example

This example contains a application-level script.. The program displays the number of layers in each open documents by using the **LayersNum** method.

```
' Loop through all documents open in the application
' and display the name and number of layers
' for each document
For i=1 To thisApp.DocsNum()
    TRACE "Document : " & thisApp.Doc(i).Name
    TRACE "    Number of layers = " & thisApp.Doc(i).LayersNum()
Next i
```

See Also	AddLayer method, Layer method, LayerByID method, LayerByName method, RemoveLayer method, RemoveLayerByID method, Layer object

Layer Method

Layer Method

Returns an instance of the **Layer** object that corresponds to the layer with the specified index in the layer collection of the document.

Applies to: Document object

Syntax

[[**Set**] *layerRet* =] *object*.**Layer** (*index*)

The **Layer** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
index	Required. An expression that returns a **Long** value. Indicates the index of the layer in the layer collection of the document.
layerRet	Optional. A **Layer** type variable.

Remarks

If *index* is less than **1** or greater than the number of layers in the layer collection of the document, the **Layer** method returns **Nothing**. Use the **LayersNum** method to find out the number of layers in the layer collection of the document. .

Example

This example contains a document-level script. The program uses the **Layer** method to go through all layers in the layer collection of the document, and displays properties of each layer.

```
' Declare variables
Dim i As Integer
Dim player As Layer
' Loops through all layers in the document
' and display each layer's properties
For i=1 To thisDoc.LayersNum()
    Set player = thisDoc.Layer(i)
    TRACE "Layer_# " & i
    TRACE "    ID = " & player.ID
    TRACE "    Name = " & player.Name
    TRACE "    Visible = " & player.Visible
    TRACE "    Locked = " & player.Locked
    TRACE "    Printable = " & player.Printable
    TRACE "    Colored = " & player.Colored
Next i
```

See Also	AddLayer method, LayerByID method, LayerByName method, LayersNum method, RemoveLayer method, RemoveLayerByID method, Layer object

LibByName Method (Application object)

LibByName Method (Application object)

Searches for a library with the specified name (**Name** property) among the open libraries of the application. Returns an instance of the **Library** object corresponding to the found library.

Applies to: Application object

Syntax

[[**Set**] *libRet* =] *object*.**LibByName** (*libName*)

The **DocByName** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the **Application** object.
libName	Required. An expression that returns a **String** value. The name (**Name** property) of the library being searched.
libRet	Optional. A **Library** type variable.

Remarks

The **LibByName** method searches for a library with the *libName* name starting from the first library in the library collection and returns the first found library. That is, if the third and fifth library have the same name, the **LibByName** method will returns the instance of the **Library** object that corresponds to the third library. If there is no matching library, the method returns **Nothing**.

LibByName Method (Window object)

LibByName Method (Window object)

Returns a **Library** object by the library name (the **Name** property).

Applies to: Window object

Syntax

[[**Set**] *libraryRet* =] *object*.**LibByName** (*libraryName*)

The **LibByName** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Window** object .
libraryName	Required. An expression that returns a **String** value. The name of the library to be found.
libraryRet	Optional. A **Library** type variable.

Remarks

This method is only effective if the window is a library window (see the **Type** property). For window of all other types the **LibByName** method always returns **Nothing**.

The **LibByName** method searches for the library with the specified name (the **Name** property) starting from the beginning of the library list of the window, and returns the first library found. If there is no library with such name in the collection, the **LibByName** method returns **Nothing**.

See Also Name property, Type property, FindLib method, Lib method, LibsNum method, Library object

LibsNum Method (Application object)

LibsNum Method (Application object)

Returns the number of open libraries in the application.

Applies to: Application object

Syntax

[[**Set**] *countRet* =] *object*.**LibsNum** ()

The **LibsNum** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the **Application** object.
countRet	Optional. A **Long** type variable.

Remarks

It's convenient to use the **LibsNum** method together with the **Lib** method to go through the open libraries in the application.

LibsNum Method (Window object)

LibsNum Method (Window object)

Returns the number of open libraries in the library window.

Applies to: Window object

Syntax

[[**Let**] *countRet* =] *object*.**LibsNum** ()

The **LibsNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Window** object.
countRet	Optional. A **Long** type variable.

Remarks

This method is only effective if the window is a library window (see the **Type** property). For window of all other types the **LibsNum** method always returns **Nothing**.

The **LibsNum** always returns a value equal to or greater than **1**, as ConceptDraw automatically closes the library window when the last library in the window has been closed.

See Also	Type property, FindLib method, Lib method, LibByName property, Library object

LibWindowByID Method

LibWindowByID Method

Searches for a library window with the specified ID (the **ID** property) in the window collection of the application. Returns an instance of the **Window** object, that corresponds to the found library window.

Applies to: Application object

Syntax

[[**Set**] *libWndRet* =] *object*.**LibWindowByID** (*libWindowID*)

The **LibWindowByID** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Application** object.
libWindowID	Required. An expression that returns a **Long** value. The ID of the library window being searched.
libWndRet	Optional. A **Window** type variable.

Remarks

If there is no library window with the specified *libWindiwID* in the window collection of the document, the **LibWindowByID** method returns **Nothing**.

Example

This example contains a application-level script. The *isLibWnd* function employs the **LibWindowByID** method and is used to determine whether the specified window is a library window.

```
' The function checks whether the window with the specified ID
' is a library window.
Function isLibWnd( wnd As Window )
    if thisApp.LibWindowByID( wnd.ID ) <> Null Then
        isLibWnd = True
    Else
        isLibWnd = False
    End If
End Function
' Declare variables
Dim l_wnd As Window
Dim d_wnd As Window
' Get first library window
Set l_wnd = thisApp.FirstLibWindow()
' Get second library window
Set d_wnd = thisApp.Doc(1).FirstView()
```

```
' Disply the results of
' the isLibWnd function for l_wnd and d_wnd
TRACE isLibWnd( l_wnd )
TRACE isLibWnd( d_wnd )
```

At least one library and one documents must be open for this example to work correctly. As the result, the following will be displayed:

```
TRUE
FALSE
```

See Also ID property, FirstLibWindow method, LibWindowsNum method, NextLibWindow method, Window object

LibWindowsNum Method

LibWindowsNum Method

Returns the number of library windows, open in the application.

Applies to: Application object

Syntax

[[**Let**] *countRet* =] *object*.**LibWindowsNum** ()

The **LibWindowsNum** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Application** object.
countRet	Optional. A **Long** type variable.

Remarks

If there are no library windows in the application, the **LibWindowsNum** method returns **0**. Also note, that the number of library windows is less or equal to the number of open libraries, because a library window can't exist without a library in it. So, the following expression always returns **True**:

thisApp.LibWindowsNum() <= thisApp.LibsNum() ' returns True

Example

This example contains a application-level script. The script calculates and displays the average number of libraries in each library window.

```
TRACE thisApp.LibsNum()/thisApp.LibWindowsNum()
```

See Also FirstLibWindow method, LibWindowByID method, NextLibWindow method, Window object

Lib Method (Application object)

Lib Method (Application object)

Returns a **Library** object by its index in the library collection of the **Application**.

Applies to: Application object

Syntax

[[**Set**] *libraryRet* =] *object*.**Lib** (*index*)

The **Lib** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Application** object .
index	Required. An expression that returns a **Long** value. The library index in the library collection of the **Application**.
libraryRet	Optional. A **Library** type variable.

Remarks

If *index* is less than **1** or greater than the number of libraries in the library collection of the **Application**, the **Lib** method returns **Nothing**. To find out the number of libraries in the library collection of the **Application**. use the **LibsNum** method.

Lib Method (Window object)

Lib Method (Window object)

Returns a **Library** object by its index in the library collection of the window.

Applies to: Window object

Syntax

[[**Set**] *libraryRet* =] *object*.**Lib** (*index*)

The **Lib** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Window** object .
index	Required. An expression that returns a **Long** value. The library index in the library collection of the library window.
libraryRet	Optional. A **Library** type variable.

Remarks

This method is only effective if the window is a library window (see the **Type** property). For window of all other types the **Lib** method always returns **Nothing**.

If *index* is less than **1** or greater than the number of libraries in the library collection of the window, the **Lib** method returns **Nothing**. To find out the number of libraries in the library collection of the window. use the **LibsNum** method.

See Also	Type property, FindLib method, LibByName method, LibsNum method, Library object

LineTo Method

LineTo Method

Builds a line segment. Returns an instance of the **Shape** object, corresponding to the shape where the line has been built.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**LineTo** (*xEnd*, *yEnd*)

The **LineTo** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
xEnd	Required. An expression that returns a **Double** value. The X-coordinate of the end point of the line.
yEnd	Required. An expression that returns a **Double** value. The Y-coordinate of the end point of the line.
shapeRet	Optional. A **Shape** type variable.

Remarks

The line is based on two points: the begin point of the line and the end point of the line.

If *object* is a page or a group, the **LineTo** method creates the line in the current Basic shape of the page or group. If the method was called prior to the **BeginShape** method or after the **EndShape** method, the **LineTo** method doesn't create anything and returns **Nothing**.

If *object* is a simple shape, the **LineTo** method creates the line in this shape.

In any case, the begin point of the line is the end point of the last geometry of the shape, in which the segment is being built. To reposition the begin point of the line, use the **MoveTo** method. The coordinates of the points are in the coordinate system of the shape, group or the page to which *object* corresponds. The coordinates are measured in internal units (**InternalUnit**).

See Also ArcTo method, BeginShape method, EndShape method, MoveTo method, SplineStart method, SplineTo method

LPtoGP Method

LPtoGP Method

Converts the coordinate of the point from the local coordinate system of the shape into the global coordinate system of the parent object (group or page).

Applies to: Shape object

Syntax

object.**LPtoGP** (*srcPoint*)

The **LPtoGP** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
srcPoint	Required. A **DPoint** type variable. The coordinates of the point.

Remarks

The global coordinate system with respect to the coordinate system of the given shape is the local coordinate system of the parent group in the case when the shape is inside a group. If the shape is not Element of a group, the global coordinate system coincides with the global coordinate system.

This method modifies the input argument *srcPoint* and uses it to return the resulting coordinates. The coordinates are measured in internal units (**InternalUnit**).

See Also GPtoLp, LAtoWA method, LPtoWP method, WPtoLP method

LPtoWP Method

LPtoWP Method

Converts the coordinates of the specified point from the local coordinate system of this shape to the world coordinate system.

Applies to: Shape object

Syntax

object.**LPtoWP** (*srcPoint*)

The **LPtoWP** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
srcPoint	Required. A **DPoint** type variable. The coordinates of the point.

Remarks

This method modifies the input argument *srcPoint* and uses it to return the resulting coordinates. The coordinates are measured in internal units (**InternalUnit**).

See Also GPtoLp, LAtoWA method, LPtoGP method, WPtoLP method

MasterByName Method

MasterByName Method

Searches for a master object with the specified name (the **Name** property) in the master object collection of the library. Returns a **Master** object that corresponds to the found master object.

Applies to: Library object

Syntax

[[**Set**] *masterObj* =] *object*.**MasterByName** (*masterName*)

The **MasterByName** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Library** object.
masterName	Required. An expression that returns a **String** value. The name of the master object.
masterObj	Optional. A **Master** type variable.

Remarks

The **MasterByName** method searches for a master object with the specified name starting from the beginning of the master object collection of the library, and returns the first found master object. If there is no master object with the *masterName* name in the master object collection of the library, the **MasterByName** method returns **Nothing**.

See Also	AddMaster method, FindMaster method, Master method, MasterByName method, MastersNum method, RemoveMaster method, RemoveMasterByName method

MastersNum Method

MastersNum Method

Returns the number of master objects in the library.

Applies to: Library object

Syntax

[[**Let**] *count* =] *object*.**MastersNum** ()

The **MastersNum** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns a **Library** object.
count	Optional. A **Long** type variable.

Remarks

If there are no master objects in the library, the **MastersNum** method returns **0**. It's convenient to use this method together with the **Master** method to go through all master objects in a library.

See Also	AddMaster method, FindMaster method, Master method, MasterByName method, MastersNum method, RemoveMaster method, RemoveMasterByName method

Master Method

Master Method

Returns an instance of the **Master** object corresponding to a master object with the specified index in the master object collection of the library.

Applies to: Library object

Syntax

[[**Set**] *masterObj* =] *object*.**Master** (*index*)

The **Master** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns a **Library** object.
index	Required. An expression that returns a **Long** value. Represents the index of the master object.
masterObj	Optional. A **Master** type variable.

Remarks

If *index* is less than **1** or greater than the number of master objects in the library, the **Master** method returns **Null**. The number of the master objects in the library can be found out with the **MastersNum** method. The inverse method to this one is the **FindMaster** method, which returns the index of the master object in the master object collection of the library.

See Also	AddMaster method, FindMaster method, Master method, MasterByName method, MastersNum method, RemoveMaster method, RemoveMasterByName method

Maximize Method

Maximize Method

Maximizes the window to full screen.

Applies to: Window object

Syntax

object.**Maximize** ()

The **Maximize** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns a **Window** object.

Remarks

Use the **State** property to find out the current state of the window.

See Also State property, Minimize method, Restore method

MenuItemByCmdID Method

MenuItemByCmdID Method

Returns an instance of the **MenuItem** object by the specified ID of the menu item (the **CmdID** property) from the menu item collection of the menu.

Applies to: Menu object

Syntax

[[**Set**] *menuItemRet* =] *object*.**MenuItemByCmdID** (*mItemCmdID*)

The **MenuItemByCmdID** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Menu** object.
mItemCmdID	Required. An expression that returns a **Long** value. Represents the ID of the menu item being searched.
menuItemRet	Optional. A **MenuItem** type variable.

Remarks

If there is no menu item with the specified *mItemCmdID*, the **MenuItemByCmdID** method returns **Nothing**.

MenuItemsNum Method

MenuItemsNum Method

Returns the number of the menu items, contained in the menu.

Applies to: Menu object

Syntax

[[**Let**] *menuItemsNumRet* =] *object*.**MenuItemsNum** ()

The **MenuItemsNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Menu** object.
menuItemsNumRet	Optional. A **Long** type variable.

Remarks

If there are no items in the menu, the **MenuItemsNum** method returns **0**.

See Also AddMenuItem method, MenuItem method, MenuItem object

MenuItem Method

MenuItem Method

Returns an instance of the **MenuItem** object by its index in the menu item collection of the menu.

Applies to: Menu object

Syntax

[[**Set**] *menuItemRet* =] *object*.**MenuItem** (*index*)

The **MenuItem** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Menu** object.
index	Required. An expression that returns a **Long** value. Indicates the index of the menu item in the menu item collection of the menu.
menuItemRet	Optional. A **MenuItem** type variable.

Remarks

If *index* is less than **1** or greater than the number of the items in the menu, the **MenuItem** method returns **Null**. Use the **MenuItemsNum** method to find out the number of menu items in the menu.

See Also MenuItemsNum method, MenuItem object

Minimize Method

Minimize Method

Minimizes the window.

Applies to: Window object

Syntax
object.**Minimize** ()

The **Minimize** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns a **Window** object.

Remarks

Use the **State** property to find out the current state of the window.

See Also State property, Maximize method, Restore method

MoveShapeToGroup Method

MoveShapeToGroup Method

Moves the object (shape) at a specific position in the specified group.

Applies to: Document object

Syntax

object. **MoveShapeToGroup** *(shp, gr, x, y, Place)*

The **MoveShapeToGroup** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an **Document** object.
shp	Required. An expression that returns a value of type **Shape.** Relocatable object.
gr	Required. An expression that returns a value of type **Shape.** The object-group, which moves the specified object.
x	Required. An expression that returns a **Double** value, representing the X coordinate of the point.
y	Required. An expression that returns a **Double** value, representing the Y coordinate of the point.
Place	Optional. An expression that returns a value of type **Long.** The position of the object in the collection of objects of the group.

Remarks

If the position of the *Place* is not specified, the default, the object is placed at the end of a collection of objects.

Methods **MoveShapeToGroup ()** and **MoveShapeToPage ()** are used to move an object (shape) in a group or to another page, respectively. To move within the same group or page using the properties of the object **GPinY GPinX** and **Shape.**

See Also	MoveTo Method, GPinX Property, Property GPinY, MoveShapeToPage () Method, Shape object, Document object

MoveShapeToPage Method

Moves the object (shape) in a specific location on that page of the document.

Applies to: Document object

Syntax

object. **MoveShapeToPage** *(shp, pg, x, y, Place)*

The **MoveShapeToPage** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an **Document** object.
shp	Required. An expression that returns a value of type **Shape.** Relocatable object.
pg	Required. An expression that returns a value of type **Page.** Page that moves the specified object.
x	Required. An expression that returns a **Double** value, representing the X coordinate of the point.
y	Required. An expression that returns a **Double** value, representing the Y coordinate of the point.
Place	Optional. An expression that returns a value of type **Long.** The position of the object in the collection of objects of the group.

Remarks

If the position of the *Place* is not specified, the default, the object is placed at the end of a collection of objects.

Methods **MoveShapeToGroup ()** and **MoveShapeToPage ()** are used to move an object (shape) in a group or to another page, respectively. To move within the same group or page using the properties of the object **GPinY GPinX** and **Shape.**

See Also	MoveTo method, GPinX property, GPinY property, MoveShapeToGroup() method, Shape object, Document object

MoveTo Method

MoveTo Method

Sets the position of the current point of the shape, used for creating the shape.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**MoveTo** (*x*, *y*)

The **MoveTo** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
x	Required. An expression that returns a **Double** value, representing the X coordinate of the point.
y	Required. An expression that returns a **Double** value, representing the Y coordinate of the point.
shapeRet	Optional. A **Shape** type variable.

Remarks

If *object* is a page or a group, the **MoveTo** method adds a new geometry with one start segment with the specified coordinates to the current Basic shape of the page or group. If the method was called before the **BeginShape** method or after the **EndShape** method, the **MoveTo** method doesn't create anything and returns **Nothing**.

If *object* is a simple shape, the **MoveTo** method adds a new geometry with the start segment in this shape.

The coordinates of the point are specified in the coordinate system of the shape, group or page to which the instance of *object* corresponds. The units of measure for the coordinates are the internal units (**InternalUnit**).

See Also ArcTo method, BeginShape method, EndShape method, LineTo method, SplineStart method, SplineTo method

NextDoc Method

NextDoc Method

Returns an instance of the **Document** object corresponding to the next document in the document collection of the application.

Applies to: Application object

Syntax

[[**Set**] *documentRet* =] *object*.**NextDoc** ()

The **NextDoc** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Application** object.
documentRet	Optional. A **Document** type variable.

Remarks

Note, that **FirstDoc** method must be called prior to using the **NextDoc** method, otherwise the **NextDoc** method will return **Nothing**. The **NextDoc** method also returns **Nothing** when the end of the list of the open documents is reached. It's convenient to use the **NextDoc** method together with the **FirstDoc** method to go through all documents open in the application.

Example

This example contains a application-level script. The script adds a page in each other document, open in the application. The **FirstDoc** and **NextDoc** methods are used to go through document.

```
Dim curDoc As Document            ' Declare variables
Set curDoc = thisApp.FirstDoc()   ' Get first document
While curDoc <> Nothing           ' Loop through all documents
    curDoc = thisApp.NextDoc()    ' Get next document
    If curDoc <> Nothing Then
        curDoc.AddPage()          ' Add page
        thisApp.NextDoc()         ' Skip next document
    End If
Wend
```

See Also	CloseDoc method, CreateNewDoc method, Doc method, DocByName method, DocsNum method, FirstDoc method, OpenDoc method, Document object

NextLibWindow Method

NextLibWindow Method

Returns an instance of the **Window** object corresponding to the next library window in the window collection of the application.

Applies to: Application object

Syntax

[[**Set**] *libWindowRet* =] *object*.**NextLibWindow** ()

The **NextLibWindow** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Application** object.
libWindowRet	Optional. A **Window** type variable.

Remarks

Note, that **FirstLibWindow** method must be called prior to using the **NextLibWindow** method, otherwise the **NextLibWindow** method will return **Nothing**. The **NextLibWindow** method also returns **Nothing** when the end of the list of the library windows has been reached. It's convenient to use the **NextLibWindow** method together with the **FirstLibWindow** method to go through all library winodws open in the application.

Example

This example contains a application-level script. The script displays the list of library windows and the list of libraries in each window.

```
' Declare variables
Dim lib_wnd As Window
Dim i As Integer
' Get the first library window
Set lib_wnd = thisApp.FirstLibWindow()
TRACE "================================"
While lib_wnd <> Null
    ' Display the ID of the window
    TRACE "ID = " & lib_wnd.ID & " :"
    ' Display the list of libraries in the current library window
    For i=1 To lib_wnd.LibsNum()
        TRACE "            - " & lib_wnd.Lib(i).Title
    Next i
    ' Get next library window
    Set lib_wnd = thisApp.NextLibWindow()
Wend
TRACE "================================"
```

See Also FirstLibWindow method, LibWindowByID method, LibWindowsNum method, Window object

NextView Method

NextView Method

Returns an instance of the **Window** object corresponding to the next window in the window collection of the document.

Applies to: Document object

Syntax

[[**Set**] *windowRet* =] *object*.**NextView** ()

The **NextView** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns a **Document** object.
windowRet	Optional. A **Window**.type variable.

Remarks

The **FirstView** method must be called prior to using the **NextView** method, otherwise the **NextView** method will return **Nothing**. The **NextView** method also returns **Nothing** when the end of the list of the windows has been reached.

Note, that the window collection of the document can include windows of the following types (the **Type** property): document window, table window, Basic window. It's convenient to use the **NextView** method together with the **FirstView** method to go through all windows of the document.

See Also Type property, FirstView method, ViewByID method, ViewsNum method, UpdateAllViews method, Window object

NormalizeRect Method

NormalizeRect Method

Normalizes the properties of the instance of the object. That means that when the left value is greater than the right value, the values are exchanged. The same applies to the top and bottom properties, if the top value is greater than bottom.

Applies to objects: DRect

Syntax

object.**NormalizeRect** ()

The **NormalizeRect** method syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.

Example

```
Dim MyRect as new DRect
MyRect.SetRect(400,400,200,200)
' After this method has been used
' MyRect properties become 200,200,400,400
MyRect.NormalizeRect()
```

See Also DRect Object

OffsetRect Method

OffsetRect Method

Offsets the rectangle by the X and Y axes, calcualtes new coordinates for the instance of the object.

Applies to objects: DRect

Syntax

object.**OffsetRect** (*x*, *y*)

The **OffsetRect** method syntax has these Elements:

Element	Description
object	A reference to an instance of the object.
x	A Double value, represents the horizontal offset of the rectangle.
y	A Double value, represents the vertical offset of the rectangle.

Example

```
' Create a new instance of the object
Dim MyObject as new DRect
' set left,top,right,bottom properties
MyObject.SetRect(100,100,200,300)
' offset rect
' After the operation the properties of MyObject will be equal to:
' left - 110, top - 85, right - 210, bottom - 285
MyObject.OffsetRect(10,-15)
```

See Also DRect Object

OpenDoc Method

OpenDoc Method

Opens an existing ConceptDraw document.

Applies to: Application object

Syntax

[[**Set** *documentRet* =] *object*.**OpenDoc** (*fileName*)

The **OpenDoc** method syntax has these Elements:

Element	Description

object	Required. An expression, that returns an instance of the **Document** object.
fileName	Required. An expression that returns a **String** value. Contains the name of the document file including the full or relative path.
documentRet	Optional. A **Document** type variable.

Remarks

The **OpenDoc** method attempts to open the specified file as one of the ConceptDraw files, to which the following file formats belong:

Extention	**File Type (for Mac)**	**CDBasic constant**	**Description**
"cdd"	'cdda'	cdf_CDD	ConceptDraw V Document
"cdd"	'cdda'	cdf_CDD1X	ConceptDraw 1.x Document
"cdl"	'cddc'	cdf_CDL	ConceptDraw V Library
"cdl"	'cddc'	cdf_CDL1X	ConceptDraw 1.x Library
"cdx"		cdf_CDX	XML for ConceptDraw
"cdw"	'cddd'	cdf_CDW	ConceptDraw V Workspace
"cdw"	'cddd'	cdf_CDW1X	ConceptDraw 1.x Workspace

If the document has been opened successfully, the **OpenDoc** method returns an instance of the **Document** object, that corresponds to the opened document. If the file with the specified name doesn't exist, or doesn't match one of the file formats, listed above, the **OpenDoc** method returns **Nothing**.

Example

This example contains a application-level script. The script imitates the way the "File->Open" menu item of ConceptDraw works.

```
Dim str As String           ' Declare string variable
str = GetOpenFileName ()    ' Display file open dialog
                            ' and get the filename
thisApp.OpenDoc( str )      ' Attempt to open the chosen file
```

See Also CloseDoc method, CreateNewDoc method, Doc method, DocByName method, DocsNum method, FirstDoc method, NextDoc method, Document object, Import/Export constants

OpenLib Method

Opens an existing ConceptDraw library. Returns an instance of the **Library** object, that corresponds to the opened library.

Applies to: Application object

Syntax

[[**Set**] *libRet* =] *object*.**OpenLib** (*fileName*)

The **OpenLib** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Application** object.
fileName	Required. An expression that returns a **String** value. Contains the name of the library file including the full or relative path.
libRet	Optional. A **Library** type variable.

Remarks

The **OpenLib** method is used for opening only library files - both in ConceptDraw V and ConceptDraw 1.x format. If the file with the specified *fileName* doesn't exist, or isn't a ConceptDraw library, the **OpenLib** method returns **Nothing**.

Example

This example contains a application-level script. It opens the library, chosen by the user, then creates a new document and copies all shapes from the library to to first page of the document.

```
Dim pLib As Library              ' Declare variables
Dim pDoc As Document
Dim lib_name As String
' Get filename from the user
Set lib_name = GetOpenFileName()
' Open the library with specified filename
Set plib = thisApp.OpenLib( lib_name )
If plib <> Null Then
    ' Create new document
    Set pDoc = thisApp.CreateNewDoc()
    ' Copy all shapes from the opened library to
    ' the first page of the document
    For i=1 To plib.MastersNum()
        pDoc.Page(1).DropStamp( plib.Master(i).Shape, 700, 600 )
    Next i
End If
```

See Also CloseLib method, CreateNewLib method, FindLib method, Lib method, LibByName method, LibsNum method, OpenLib method, Library object

OpenWorkspace Method

OpenWorkspace Method

Opens an existing workspace file.

Applies to: Application object

Syntax

[[**Let**] *booleanRet* =] *object*.**OpenWorkspace** (*fileName*)

The **OpenWorkspace** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Application** object.
fileName	Required. An expression that returns a **String** value. Contains the name of the workspace file including the full or relative path.
booleanRet	Optional. A **Boolean** type variable.

Remarks

If the specified file has been successfully found and opened as a ConceptDraw workspace file, the **OpenWorkspace** method returns **True**. Otherwise, the method returns **False**. To open ConceptDraw documents and libraries separately use the **OpenDoc** and **OpenLib** methods respectively.

Example

This example contains a application-level script. The script uses the **OpenWorkspace** method to imitate the way the "File->Open Workspace" menu item of ConceptDraw works.

```
thisApp.OpenWorkSpace( GetOpenFileName() )
```

See Also OpenDoc method, OpenLib method, SaveWorkspace method

PageByID Method

PageByID Method

Searches for a page by the specified ID (the **ID** property) in the page collection of the document. Returns an instance of the **Page** object, corresponding to the found page.

Applies to: Document object

Syntax

[[**Set**] *pageRet* =] *object*.**PageByID** (*pageID*)

The **PageByID** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Document** object.
pageID	Required. An expression that returns a **Long** value. Represents the **ID** of the page being searched.
pageRet	Optional. A **Page** variable.

Remarks

If the page with the specified *pageID* wasn't found in the document, the method returns **Nothing**.

Example

This example contains a document-level script. The script removes all shapes on the page, specified by the user. The user specifies the ID of the page, then the **PageByID** method is used to find the page and the script removes all shapes and service objects on the page.

```
' Declare variables
Dim ppage As Page     ' page
Dim pageID As Long    ' Page ID
' Ask the user to input page ID
Let pageID = InputBox( "Enter ID of page :", "CLEAR UP Page!" )
' Get the reference to the page by the page ID
' provided by the user
Set ppage = thisDoc.PageByID( pageID )
' If the page with such ID was found
' in the page collection of the document, clear the page
If ppage <> Null Then
    ' Remove all shapes on the page
    ppage.RemoveAllShapes()
    ' Remove all service objects on the page
    ppage.RemoveAllServObjs()
    ' Display a message that the page has been cleaned up
    MsgBox( "Page is clear!" )
Else
    ' Display a message the the page has not been found
    MsgBox( "No page was chosen!" )
End If
```

See Also	AddPage method, FindPage method, Page method, PagesNum method, RemovePage method, RemovePageByID method, ReorderPage method, ReorderPageByID method, Page object

PagesNum Method

PagesNum Method

Returns the number of the pages in the page collection of the document.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**PagesNum** ()

The **PagesNum** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Document** object.
countRet	Optional. A **Long** type variable.

Remarks

Returns **1** or greater, as there's always at least one page in the document.

Example

This example contains a application-level script.. The program displays list of documents, open in the application, and the number of pages in each document by using the **PagesNum** method.

```
' Declare variables
Dim pdoc As Document
' Loop through all open documents
For i=1 To thisApp.DocsNum()
    ' Get next document
    Set pdoc = thisApp.Doc(i)
    ' Display the name of the document
    TRACE "Document Name : " & pdoc.Name
    ' Display the number of pages in the document
    TRACE "    Number of pages = " & pdoc.PagesNum()
Next i
```

See Also AddPage method, FindPage method, Page method, PageByID method, RemovePage method, RemovePageByID method, ReorderPage method, ReorderPageByID method, Page object

Page Method

Page Method

Returns an instance of the **Page** object that corresponds to the page with the specified index in the page collection of the document.

Applies to: Document object

Syntax

[[**Set**] *pageRet* =] *object*.**Page** (*index*)

The **Page** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Document** object.
index	Required. An expression that returns a **Long** value. Indicates the index of the page in the page collection of the document.
pageRet	Optional. A **Page** type variable.

Remarks

If *index* is less than **1** or greater than the number of pages in the page collection of the document, the **Page** method returns **Nothing**. Use the **PagesNum** method to find out the number of pages in the page collection of the document. .

Example

This example contains a document-level script. The program displays all page names and properties of each page in the page collection of the document. The **Page** method is used to go through all pages in the collection.

```
' Declare variables
Dim ppage As Page
' Loop through all pages of the document
' by using the Page method
For i=1 To thisDoc.PagesNum()
    ' Get next page from the pae collection
    ' of the document
    Set ppage = thisDoc.Page(i)
    ' Display the page index
    TRACE "Page_#_" & i
    ' display a  page property
    TRACE "    ID = " & ppage.ID
    TRACE "    Name = " & ppage.Name
    TRACE "    isBackground = " & ppage.isBackground
    TRACE "    BackPageID = " & ppage.BackPageID
Next i
```

See Also AddPage method, FindPage method, PageByID method, PagesNum method, RemovePage method, RemovePageByID method, ReorderPage method, ReorderPageByID method, Page object

ParagraphsNum Method

ParagraphsNum Method

Returns the number of paragraphs in the paragraph collection of the shape.

Applies to: Shape object

Syntax

[[**Let**] *countRet* =] *object*.**ParagraphsNum** ()

The **ParagraphsNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
countRet	Optional. A **Long** type variable.

Remarks

If the shape doesn't contain text, it doesn't contain any paragraphs, so in this case the **ParagraphsNum** method returns **0**.

See Also GetParagraphIndex method, Paragraph method, RemoveParagraph method, SetParaAfterSpacing method, SetParaBeforeSpacing method, SetParaFirstInd method, SetParaHAlign method, SetParaLeftInd method, SetParaLineSpacing method, SetParaRightInd method, Paragraph object

Paragraph Method

Paragraph Method

Returns a **Paragraph** object that corresponds to a paragraph with the specified index in the paragraph collection of the shape.

Applies to: Shape object

Syntax

[[**Set**] *paragraphRet* =] *object*.**Paragraph** (*index*)

The **Paragraph** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the paragraph in the paragraph collection of the shape.
paragraphRet	Optional. A **Paragraph** type variable.

Remarks

If *index* is less than **1** or greater than the number of paragraphs of the shape, the **Paragraph** method returns **Nothing**. To find out the number of paragraphs of the shape, use the **ParagraphsNum** method.

See Also GetParagraphIndex method, ParagraphsNum method, RemoveParagraph method, SetParaAfterSpacing method, SetParaBeforeSpacing method, SetParaFirstInd method, SetParaHAlign method, SetParaLeftInd method, SetParaLineSpacing method, SetParaRightInd method, Paragraph object

PropertyChanged Method

PropertyChanged Method

Tells ConceptDraw engine that the value of the specified property has been changed and the formulas of the dependent properties must be re-calculated.

Applies to objects: Shape, ServObj

Syntax

object.**PropertyChanged**(*propTag* [, *num*[, *geom*]])

The **PropertyChanged** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object and **ServObj** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag, num, geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

If the **PropertyChanged** method has been called after the **StartRebuild** method, the properties will be re-calculated on calling the **EndRebuild** method. Otherwise, they will be re-calculated immediately.

If the **PropertyChanged** method has been called from a user procedure, which in its turn has been called during re-calculation of a property, containing the table formula with functions _CALLTHIS, _CALLTHIS_1ARG or _CALLTHIS_2ARGS, the properties, depending on the property specified in **PropertyChanged** will be re-calculated as soon as the calculation of the property that called the user procedure is over.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method , EndRebuild method, StartRebuild method

PtInRect Method

PtInRect Method

Returns a Boolean value: TRUE, if the point with the specified coordinates is within the rectanlge, otherwise - FALSE.

Applies to objects: DRect

Syntax

[[**Let**] *res* =] *object*.**PtInRect** (*x*, *y*)

The **PtInRect** method syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.
x, *y*	X,Y coordinates of the point, Double values.
res	A Boolean type variable.

Example

This example demonstrates using the PtInRect method.

```
Dim MyObject as new DRect, res as Boolean
'set DRect object properties
MyObject.SetRect(100,100,200,500)
res = MyObject.PtInRect(150,250)    ' res = true
```

See Also DRect Object

RecalcProperty Method

RecalcProperty Method

Tells ConceptDraw engine that the value of the specified property is to be re-calculated using its table formula.

Applies to objects: Shape, ServObj

Syntax

object.**RecalcProperty**(*propTag* [, *num*[, *geom*]])

The **RecalcProperty** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the shape.
num	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the shape.
geom	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the shape.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object and **ServObj** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag, num, geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

If the **RecalcProperty** method has been called after the **StartRebuild** method, the properties will be re-calculated on calling the **EndRebuild** method. Otherwise, they will be re-calculated immediately.

If the **RecalcProperty** method has been called from a user procedure, which in its turn has been called during re-calculation of a property, containing the table formula with functions _CALLTHIS, _CALLTHIS_1ARG or _CALLTHIS_2ARGS, the properties, depending on the property specified in **RecalcProperty** will be re-calculated as soon as the calculation of the property that called the user procedure is over.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method , EndRebuild method, StartRebuild method

RemoveAction Method

RemoveAction Method

Removes a user-defined action with the specified index from the user-defined action collection of the shape, and returns the number of remaining actions.

Applies to: Shape object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveAction** (*index*)

The **RemoveAction** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the user-defined action to be deleted.
actionRet	Optional. An **Action** type variable.

Remarks

If *index* is less than **1** or greater than the number of user-defined actions of the shape, the **RemoveAction** method doesn't delete anything and returns the number of user-defined actions of the shape. To find out the number of user-defined actions of the shape, use the **ActionsNum** method.

See Also Action method, ActionsNum method, AddAction method, RemoveAction method

RemoveAllServObjs Method

RemoveAllServObjs Method

Removes all service objects that belong to the page/group.

Applies to: Page object, Shape object

Syntax

object.**RemoveAllServObjs** ()

The **RemoveAllServObjs** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.

Remarks

The **RemoveAllServObjs** method attempts to remove all service objects of the page/group. To find out whether all service objects have been removed, use the **ServObjsNum** method which returns the number of service objects, that belong to the page/group.

See Also	RemoveAllShapes method, RemoveServObj method, RemoveServObjByID method, ServObjsNum method

RemoveAll Method

RemoveAllShapes Method

Removes all shapes that belong to the page/group.

Applies to: Page object, Shape object

Syntax

object.**RemoveAllShapes** ()

The **RemoveAllShapes** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.

Remarks

The **RemoveAllShapes** method attempts to remove all shapes of the page/group. Note, that you can't delete the shape which Basic script is being executed at the moment. To find out whether all shapes have been removed, use the **ShapesNum** method which returns the number of shapes, that belong to the page/group.

See Also	RemoveAllServObj method, RemoveShape method, RemoveShapeByID method, ShapesNum method

RemoveAll Method

Removes all menu items from the menu.

Applies to: Menu object

Syntax
object.**RemoveAll** ()

The **MenuItem** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Menu** object.

RemoveCharacter Method

RemoveCharacter Method

Removes the character block with the specified index from the character block collection of the shape, and returns the number of remaining character blocks.

Applies to: Shape object

Syntax
[[**Let**] *countRet* =] *object*.**RemoveCharacter** (*index*)

The **RemoveCharacter** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the character block to be deleted.
indexRet	Optional. A **Long** type variable.

Remarks

If *index* is less than **1** or greater than the number of character blocks of the shape, the **RemoveCharacter** method doesn't delete anything and returns the current number of character blocks of the shape. To find out the number of character blocks of the shape, use the **CharactersNum** method.

See Also	Character method, CharactersNum method, GetCharacterIndex method, SetCharColor method, SetCharFont method, SetCharHyperlink method, SetCharLanguage method, SetCharPos method, SetCharSize method, SetCharSpacing method, SetCharStyle method, Character object

RemoveConnectDot Method

RemoveConnectDot Method

Removes a connection point from the connection point collection of the shape, and returns the number of remaining connection points.

Applies to: Shape object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveConnectDot** (*index*)

The **RemoveConnectDot** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the connection point to be deleted from the connection point collection of the shape.
connectDotRet	Optional. A **ConnectDot** type variable.

Remarks

If *index* is less than **1** or greater than the number of connection points of the *object* shape, the **RemoveConnectDot** method doesn't delete anything and returns the current number of connection points of the shape. To find out the number of connection points of the shape, use the **ConnectDotsNum** method.

See Also	AddConnectDot method, ConnectDot method, ConnectDotsNum method, ConnectDot object

RemoveControlDot Method

RemoveControlDot Method

Removes a control handle with the specified index from the control handle collection of the shape, and returns the number of remaining control handles.

Applies to: Shape object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveControlDot** (*index*)

The **RemoveControlDot** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the control handle to be deleted in the control handle collection of the shape.
countRet	Optional. A **Long** type variable.

Remarks

If *index* is less than **1** or greater than the number of control handles of the *object* shape, the **RemoveControlDot** method doesn't delete anything and returns the current number of control handles of the shape. To find out the number of control handles of the shape, use the **ControlDotsNum** method.

See Also AddControlDot method, ControlDot method, ControlDotsNum method, ControlDot object

RemoveCustomProp Method

RemoveCustomProp Method

Removes a custom property with the specified index from the custom property collection of the shape, and returns the number of remaining custom properties.

Applies to: Shape object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveCustomProp** (*index*)

The **RemoveCustomProp** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the custom property to be deleted in the custom property collection of the shape.
countRet	Optional. A **Long** type variable.

Remarks

If *index* is less than **1** or greater than the number of custom properties of the shape, the **RemoveCustomProp** method doesn't delete anything and returns the current number of custom properties of the shape. To find out the number of custom properties of the shape, use the **CustomPropsNum** method.

See Also	AddCustomProp method, CustomProp method, CustomPropByLabel, CustomPropsNum method, CustomProp object

RemoveDataSource Method

RemoveDataSource Method

Deletes the data source from the collection of data sources, the object (shape) of the index. Returns the number of data sources in the object (shape).

Applies to: Shape object

Syntax

[[Let] *num* =] *object.* **RemoveDataSource** *(index)*

The **RemoveDataSource** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.

index	Required. An expression that returns a **Long** value. The index of the DataSource in the DataSources collection of the shape.
num	Optional. A **Long** type variable.

Remarks

The numbering of the indices of data sources in the collection begins with 1. **RemoveDataSource** method returns the total number of data sources, regardless of whether they are valid or not. If the object has no data sources, the function returns 0.

Example

```
dim num as Integer
num = thisShape.DataSourcesNum ()
trace num
num = thisShape.RemoveDataSource (1)
trace num
```

See Also DataSource object , AddDataSource Method , Method DataSource , Method DataSourcesNum

RemoveDSValue Method

RemoveDSValue Method

Removes a row from a table Data parameters of the object (shape) of the index. Returns the number of rows in a table Data parameters of the object (shape).

Applies to: Shape object

Syntax

[[Let] *num* =] *object*. **RemoveDSValue** *(index)*

The **RemoveDSValue** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the row Data parameters of the object (shape).
num	Optional. A **Long** type variable.

Remarks

The numbering of the indices of rows in a table Data begins at 1. If the object does not have a table Data parameters of the object (shape), or has been removed the last line, the method returns 0**RemoveDSValue.**

Example

```
dim num as Integer
num = thisShape.DSValuesNum()
trace num
num = thisShape.RemoveDSValue(1)
trace num
```

See Also DataSourceValue object, AddDSValue method, DSValue method, DSValueEl method, DSValuesNum method

RemoveGeometry Method

RemoveGeometry Method

Removes a geometry with the specified index from the geometry collection of the shape, and returns the number of remaining geometries.

Applies to: Shape object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveGeometry** (*index*)

The **RemoveGeometry** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the geometry in the geometry collection of the shape.
countRet	Optional. An **Long** type variable.

Remarks

If *index* is less than **1** or greater than the number of geometries of the shape, the **RemoveGeometry** method doesn't delete anything and returns the current number of geometries of the shape. To find out the number of geometries of the shape, use the **GeometriesNum** method.

See Also AddGeometry method, GeometriesNum method, Geometry method, Geometry object

RemoveLayerByID Method

RemoveLayerByID Method

Removes the layer with the specified ID (ID property) from the layer collection of the document. Returns the number of layers, remaining in the collection after the operation.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveLayerByID** (*layerID*)

The **RemoveLayerByID** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
layerID	Required. An expression that returns a **Long** value. Indicates the ID of the layer in the layer to be removed.
countRet	Optional. A **Long** type variable.

Remarks

If the layer with the specified ID (the ID property) wasn't found, the **RemoveLayerByID** method doesn't remove anything and returns the number of layers in the document. When a layer is removed, the remaining layers are re-indexed - that is, the index of every layer after the removed one is decreased by **1**.

You can't remove all layers - at least one layer must exist in the document. An attempt to delete the last layer will have no effect.

Example

This example contains a document-level script. The program removes all layers that don't have shapes on them from the layer collection of the document. First it determines the IDs of all layers and records them into the *layerIDs* array. Then the IDs of the layers which have shapes on them are erased from the array. The layers with remaining IDs are removed by using the **RemoveLayerByID** method. The number of removed layers is calculated and displayed on the screen.

```
' Declare variables
Dim ppage As Page          ' Page
Dim player As Layer        ' Layer
Dim layers_num As Integer ' Number of layers
Dim layerIDs() As Long     ' Layer IDs array
' Get the number of layers
layers_num = thisDoc.LayersNum()
' Create an array to store the IDs of the layers in the document
ReDim layerIDs(layers_num)
' Fill the layerIDs array with IDs of all layers of the document
For i=1 To layers_num
    Let layerIDs(i) = thisDoc.Layer(i).ID
Next i
' Loop through all pages of the document
For i=1 To thisDoc.PagesNum()
    ' Get next page
    Set ppage = thisDoc.Page(i)
    ' Loop through all shapes on the page
    For j=1 To ppage.ShapesNum()
        ' For each shape determine the ID of the layer on which it's located
        ' and if it matches the currently used layer,
        ' erase it
        For k=1 To layers_num
            If layerIDs(k) = ppage.Shape(j).Layer Then
                layerIDs(k) = 0
            End If
        Next k
    Next j
Next i
layers_num = 0
' Loop through all remaining layer IDs
' and remove corresponding layers
For i=1 To thisDoc.LayersNum()
    If layerIDs(i) <> 0 Then
        thisDoc.RemoveLayerByID( layerIDs(i) )
        layers_num = layers_num + 1
    End If
Next i
' Display the number of removed layers
TRACE "Number of deleted layers = " & layers_num
```

See Also Layer method, LayerByID method, LayerByName method, LayersNum method, RemoveLayer method, Layer object

RemoveLayer Method

RemoveLayer Method

Removes the layer with the specified index from the layer collection of the document. Returns the number of layers, remaining in the collection after the operation.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveLayer** (*index*)

The **RemoveLayer** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
index	Required. An expression that returns a **Long** value. Indicates the index of the layer to be removed in the layer collection of the document.
countRet	Optional. A **Long** type variable.

Remarks

If *index* is less than **1**, or greater than the number of layers in the layer collection of the document, the **RemoveLayer** method doesn't remove the layer. When a layer is removed, the remaining layers are re-indexed - that is, the index of every layer after the removed one is decreased by **1**.

You can't remove all the layers - at least one layer must exist in the document. An attempt to delete the last layer will have no effect.

Example

This example contains a document-level script. It removes all non-printable layers of the document by using the **RemoveLayer** method.

```
' Declare variables
Dim player As Layer
' Loop through all layers
' starting from the end of the layer list of the current document
For i=thisDoc.LayersNum() To 1 Step -1
    ' Get next layer
    Set player = thisDoc.Layer(i)
    ' If layer is non-printable, remove the layer
    If player.Printable = False Then
        thisDoc.RemoveLayer(i)
    End If
Next i
```

See Also AddLayer method, Layer method, LayerByID method, LayerByName method, LayersNum method, RemoveLayerByID method, Layer object

RemoveMasterByName Method

RemoveMasterByName Method

Removes the master object with the specified name (the **Name** property) from the master object collection of the library.

Applies to: Library object

Syntax

[[**Let**] *count* =] *object*.**RemoveMasterByName** (*masterName*)

The **RemoveMasterByName** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns a **Library** object.
masterName	Required. An expression that returns a **String** value. Indicates the name (the **Name** property) of the master object to be removed.
count	Optional. A **Long** type variable.

Remarks

The **RemoveMasterByName** method searches for a master object with the specified name, starting from the beginning of the master object collection of the library, and returns the first master object, that matches the specified name. If there are no master objects with such name, the **MasterByName** method returns **Nothing**.

See Also AddMaster method, FindMaster method, Master method, MasterByName method, MastersNum method, RemoveMaster method, RemoveMasterByName method

RemoveMaster Method

RemoveMaster Method

Removes a master object with the specified index from the master object collection of the library, and returns the number of remaining master objects.

Applies to: Library object

Syntax

[[**Let**] *count* =] *object*.**RemoveMaster** (*index*)

The **RemoveMaster** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Library** object.
index	Required. An expression that returns a **Long** value. The index of the master object to be deleted.
count	Optional. A **Long** type variable.

Remarks

If *index* is less than **1**, or greater than the number of master objects in the master object collection of the library, the **RemoveMaster** method doesn't remove the master object. When a master object is removed, the remaining master objects are re-indexed - that is, the index of every master object after the removed one is decreased by **1**.

See Also	AddMaster method, FindMaster method, Master method, MasterByName method, MastersNum method, RemoveMaster method, RemoveMasterByName method

RemoveMenuItemByCmdID Method

RemoveMenuItemByCmdID Method

Removes the menu item with the specified ID (the **CmdID** property). Returns the number of menu item, remaining in the menu after the operation.

Applies to: Menu object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveMenuItemByCmdID** (*mItemCmdID*)

The **RemoveMenuItemByCmdID** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Menu** object.
mItemCmdID	Required. An expression that returns a **Long** value. Indicates the ID (the **CmdID** property) of the menu item to be removed.
countRet	Optional. A **Long** type variable.

Remarks

If the menu item with the specified *mItemCmdID* was not found in the menu item collection of the menu, the **RemoveMenuItemByCmdID** method doesn't remove anything.

RemoveMenuItem Method

RemoveMenuItem Method

Removes the menu item with the specified index Returns the number of menu item, remaining in the menu after the operation.

Applies to: Menu object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveMenuItem** (*index*)

The **RemoveMenuItem** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Menu** object.
index	Required. An expression that returns a **Long** value. Indicates the index of the menu item to be removed in the menu item collection of the menu.
countRet	Optional. A **Long** type variable.

Remarks

If *index* is less than **1** or greater than the number of menu items in the menu, the **RemoveMenuItem** method removes nothing.

See Also

RemovePageByID Method

RemovePageByID Method

Removes the page with the specified ID (the ID property) from the page collection of the document. Returns the number of pages, remaining in the document after the operation.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**RemovePageByID** (*pageID*)

The **RemovePageByID** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
pageID	Required. An expression that returns a **Long** value. Indicates the ID (the ID property) of the page to be removed in the page collection of the document.
countRet	Optional. A **Long** type variable.

Remarks

If there is no page with the specified ID (the ID property) in the document, the **RemovePageByID** method doesn't delete the page and returns the number of pages in the document. When a page is removed, the remaining pages are re-indexed - that is, the index of every page following the removed one is decreased by **1**. Use the **PagesNum** method to find out the number of the pages in the document.

You can't remove all the pages - at least one page must exist in the document. An attempt to delete the last page will have no effect. Also, you can't remove the page if its script or the script of one of its shapes is being executed at the moment.

Example

This example contains a document-level script. The program attempts to remove the page with the ID specified by the user by using the **RemovePageByID** method. Then in analyses the number of the pages in the document, and displays a message saying whether the page was removed.

```
' Declare variables
Dim pageID As Long
Dim count As Long
' The user inputs the ID of the page
' to be deleted
Let pageID = InputBox( "Enter ID of page to delete:", "Delete page by ID" )
' Remember the number of pages in the document
' before the attempt to remove the page
Let count = thisDoc.PagesNum()
' Remove the page with the specified ID
If count = thisDoc.RemovePageByID( pageID ) Then
    ' If the number of pages hasn't changed,
    ' display a message that the page wasn't deleted
    MsgBox( "Page has been deleted!" )
Else
    ' If the number of pages hast changed,
    ' display a message that the page has been  deleted
    MsgBox( "Page with ID = " & pageID & " has been deleted!" )
End If
```

See Also	AddPage method, FindPage method, Page method, PageByID method, PagesNum method, RemovePage method, ReorderPage method, ReorderPageByID method, Page object

RemovePage Method

RemovePage Method

Removes the page with the specified index from the page collection of the document. Returns the number of pages, remaining in the collection after the operation.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**RemovePage** (*index*)

The **RemovePage** method syntax has these Elements:

Element	Description

object	Required. An expression, that returns an instance of the **Document** object.
index	Required. An expression that returns a **Long** value. Indicates the index of the page to be removed in the page collection of the document.
countRet	Optional. A **Long** type variable.

Remarks

If *index* is less than **1**, or greater than the number of pages in the document, the **RemovePage** method doesn't remove the page. When a page is removed, the remaining pages are re-indexed - that is, the index of every page following the removed one is decreased by **1**. Use the **PagesNum** method to find out the number of the pages in the document.

You can't remove all the pages - at least one page must exist in the document. An attempt to delete the last page will have no effect. Also, you can't remove the page if its script or the script of one of its shapes is being executed at the moment.

Example

This example contains a document-level script. The program removes all pages that don't have shapes on them. Pages are removed with the **RemovePage** method.

```
' Loop through all pages of the document
For i=thisDoc.PagesNum() To 1 Step -1
    ' If there are no shapes on the page
    ' delete page
    If thisDoc.Page(i).ShapesNum() = 0 Then
        thisDoc.RemovePage(i)
    End If
Next i
```

See Also	AddPage method, FindPage method, Page method, PageByID method, PagesNum method, RemovePageByID method, ReorderPage method, ReorderPageByID method, Page object

RemoveParagraph Method

RemoveParagraph Method

Removes a paragraph with the specified index from the paragraph collection of the shape, and returns the number of remaining paragraphs.

Applies to: Shape object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveParagraph** (*index*)

The **RemoveParagraph** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the paragraph to be deleted in the paragraph collection of the shape.
countRet	Optional. A **Long** type variable.

Remarks

If *index* is less than **1** or greater than the number of paragraphs of the shape, the **RemoveParagraph** method doesn't delete anything and returns the current number of paragraphs of the shape. To find out the number of paragraphs of the shape, use the **ParagraphsNum** method.

See Also	GetParagraphIndex method, Paragraph method, ParagraphsNum method, SetParaAfterSpacing method, SetParaBeforeSpacing method, SetParaFirstInd method, SetParaHAlign method, SetParaLeftInd method, SetParaLineSpacing method, SetParaRightInd method, Paragraph object

RemoveServObjByID Method

RemoveServObjByID Method

Removes a service object with the specified ID (the **ID** property) from the service object collection of the group/page, and returns the number of remaining service objects.

Applies to: Page object, Shape object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveServObjByID** (*servObjID*)

The **RemoveServObjByID** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.

servObjID	Required. An expression that returns a **Long** value. The ID (the **ID** property) of the service object to be deleted.
countRet	Optional. A **Long** type variable

Remarks

If there is no service object with the specified *servObjID*, the **RemoveServObjByID** method doesn't delete anything and returns the current number of service objects in the group/page.

See Also	ID property, RemoveAllServObjs method, RemoveServObj method, ReorderServObj method, ReorderServObjByID method, ServObj method, ServObjByID method, ServObjsNum method, ServObj object

RemoveServObj Method

RemoveServObj Method

Removes a service object with the specified index from the service object collection of the group/page, and returns the number of remaining service objects.

Applies to: Page object, Shape object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveServObj** (*index*)

The **RemoveServObj** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
index	Required. An expression that returns a **Long** value. The index of the service object to be deleted in the service object collection of the group/page.
countRet	Optional. A **Long** type variable

Remarks

If *index* is less than **1**, or greater than the number of service objects in the group/page, the **RemoveServObj** method doesn't remove the service object and returns the number of service objects in the group/page. When a service object is removed, the remaining service objects are re-indexed - that is, the index of every service object after the removed one is decreased by **1**. To find out the number of service objects in the group/page, use the **ServObjsNum** method.

See Also	RemoveAllServObjs method, RemoveServObjByID method, ReorderServObj method, ReorderServObjByID method, ServObj method, ServObjByID method, ServObjsNum method, ServObj object

RemoveShapeByID Method

RemoveShapeByID Method

Removes a shape with the specified ID (the **ID** property) from the shape collection of the group/page, and returns the number of remaining shapes.

Applies to: Page object, Shape object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveShapeByID** (*shapeID*)

The **RemoveShapeByID** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
shapeID	Required. An expression that returns a **Long** value. The ID (**ID** property) of the shape to be deleted.
countRet	Optional. A **Long** type variable

Remarks

If there is no shape with the specified *shapeID* in the shape collection of the group/page, the **RemoveShapeByID** method doesn't delete anything and returns the current number of shapes in the group/page.

See Also ID property, RemoveAllShapes method, RemoveShape method, ReorderShape method, ReorderShapeByID method, Shape method, ShapeByID method, ShapesNum method

RemoveShape Method

RemoveShape Method

Removes a shape with the specified index from the shape collection of the group/page, and returns the number of remaining shapes.

Applies to: Page object, Shape object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveShape** (*index*)

The **RemoveShape** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
index	Required. An expression that returns a **Long** value. The index of the shape in the shape collection of the group/page.
countRet	Optional. A **Long** type variable.

Remarks

If *index* is less than **1**, or greater than the number of shapes in the group/page, the **RemoveShape** method doesn't remove anything and returns the current number of shapes in the group/page. When a shape is removed, the remaining shapes are re-indexed - that is, the index of every shape after the removed one is decreased by **1**. To find out the number of shapes in the group/page, use the **ShapesNum** method.

See Also RemoveAllShapes method, RemoveShapeByID method, ReorderShape method, ReorderShapeByID method, Shape method, ShapeByID method, ShapesNum method

RemoveStyleByName Method

RemoveStyleByName Method

Removes the style with the specified name (the **Name** property) from the style collection of the document. Returns the number of styles, remaining in the collection after the operation.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveStyleByName** (*styleName*)

The **RemoveStyleByName** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
styleName	Required. An expression that returns a **String** value. Indicates the name (the **Name** property) of the style to be removed.
countRet	Optional. A **Long** type variable.

Remarks

If no style with the specified *styleName* has been found, the **RemoveStyleByName** method doesn't remove the style and returns the number of styles in the style collection of the document.

See Also Name property, AddStyle method, FindStyle method, RemoveStyle method, RenameStyle method, Style method, StyleByName method, StylesNum method, Style object

RemoveStyle Method

RemoveStyle Method

Removes the style with the specified index from the style collection of the document. Returns the number of styles, remaining in the collection after the operation.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveStyle** (*index*)

The **RemoveStyle** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
index	Required. An expression that returns a **Long** value. Indicates the index of the style to be removed in the style collection of the document.
countRet	Optional. A **Long** type variable.

Remarks

If *index* is less than **1**, or greater than the number of styles in the document, the **RemoveStyle** method doesn't remove the style and returns the number of styles in the style collection of the document.

See Also	AddStyle method, FindStyle method, RemoveStyleByName method, RenameStyle method, Style method, StyleByName method, StylesNum method, Style object

RemoveTabStop Method

RemoveTabStop Method

Removes a tab stop with the specified index from the tab stop collection of the text block, and returns the number of remaining tab stops.

Applies to: TextBlock object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveTabStop** (*index*)

The **RemoveTabStop** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **TextBlock** object.
index	Required. An expression that returns a **Long** value. The index of the tab stop to be deleted.

countRet	Optional. A **Long** type variable

Remarks

If *index* is less than **1** or greater than the number of tab stops in the text block, the **RemoveTabStop** method doesn't delete anything and returns the current number of tab stops in the text block. To find out the number of tab stops in the text block, use the **TabSopsNum** method.

Example

This example contains a document-level script. It demonstrates how to delete a tab stop with number 1 in a shape. It assumes that the active page contains the shape with ID 1, which has text and at least one tab stop is defined.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Delete TabStop with  ID1, display the number of remaining tab stops
trace s.TextBlock.RemoveTabStop(1)
```

See Also AddTabStop method, TabStop method, TabStopsNum method

RemoveUnusedHyperlink Method

RemoveUnusedHyperlinks Method

Removes unused hyperlinks from the hyperlink collection of the document. Returns the number of remaining hyperlinks in the hyperlink collection of the document.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveUnusedHyperlinks** ()

The **RemoveUnusedHyperlinks** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
countRet	Optional. A **Long** type variable.

Remarks

A hyperlink is considered unused if from all shapes and characters in the document there is no one, which Hyperlink property would match the ID (the ID property) of the hyperlink. After you use this method, be careful when working with references to instances of the **Hyperlink** object, because the instances of the objects to which they referenced may no longer exist (be removed).

Example

This example contains a document-level script. It demonstrates how the **RemoveUnusedHyperlinks** is used for two hyperlinks, added by using the **AddHyperlinkToFile** method. Also it shows how an error may occur when an instance of the **Hyperlink** object, pointing to a non-existing hyperlink, is used.

```
' Declare variables
Dim hlinkID1 As Long
Dim hlinkID2 As Long
Dim shp As Shape
Dim hlink1 As Hyperlink
Dim hlink2 As Hyperlink
' Add to the hyperlink collection of the document
' two new hyperlinks to files
hlinkID1 = thisDoc.AddHyperlinkToFile( "1.cdd" )
hlinkID2 = thisDoc.AddHyperlinkToFile( "2.cdd" )
' Get the hyperlinks by their IDs
Set hlink1 = thisDoc.HyperlinkByID( hlinkID1 )
Set hlink2 = thisDoc.HyperlinkByID( hlinkID2 )
' Draw a shape and assign the first hyperlink to it
Set shp = thisDoc.ActivePage.DrawRect( 100,100,700,300 )
shp.Text = "1.cdd"
shp.Hyperlink = hlinkID1
' Remove unused hyperlinks from the hyperlink collection of the document;
' the hyperlink with ID 2 will be removed because it's not assigned to
' any object.
thisDoc.RemoveUnusedHyperlinks()
' Display the Address property
' of any of the two added hyperlinks
TRACE "Hyperlink_1 = " & hlink1.Address
' The same for the second hyperlink!
' This code can cause a run-time error, because the
' hyperlink, referenced to by the hlink2 variable,
' no longer exists
TRACE "Hyperlink_2 = " & hlink2.Address
```

See Also	ID property, Hyperlink property, AddHyperlinkToDocument method, AddHyperlinkToFile method, AddHyperlinkToPageShape method, AddHyperlinkToURL method, Hyperlink method, HyperlinkByID method, HyperlinksNum method, Hyperlink object

RemoveVariable Method

RemoveVariable Method

Removes a user-defined variable with the specified index from the user-defined variable collection of the shape, and returns the number of remaining variables.

Applies to: Shape object

Syntax

[[**Let**] *countRet* =] *object*.**RemoveVariable** (*index*)

The **RemoveVariable** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the variable to be deleted in the user-defined variable collection of the shape.
countRet	Optional. A **Long** type variable.

Remarks

If *index* is less than **1** or greater than the number of user-defined variables of the shape, the **RemoveVariable** method doesn't delete anything and returns the current number of user-defined variables of the shape. To find out the number of user-defined variables of the shape, use the **VariablesNum** method.

See Also AddVariable method, Variable method, VariablesNum method, Variable object

RenameStyle Method

RenameStyle Method

Renames a style: modifies its **Name** property.

Applies to: Document object

Syntax
[[**Let**] *booleanRet* =] *object*.**RenameStyle** (*originalStyleName*, *newStyleName*)

The **RemoveStyleByName** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
originalStyleName	Required. An expression that returns a **String** value. Represents the name of the style (the **Name** property) to be renamed.
newStyleName	Required. An expression that returns a **String** value. Represents the new name of the style.
booleanRet	Optional. A **Boolean** type variable.

Remarks

Note, that the name of the style (the **Name** property) is unique within the scope of the style collection of the document. If there is not style with the specified *originalStyleName* in the style collection, or a style with *newStyleName* already exists, the **RenameStyle** method doesn't rename the style and returns **False**. If the style was renamed successfully, this method returns **True**.

See Also	Name property, AddStyle method, FindStyle method, RemoveStyle method, RemoveStyleByName method, Style method, StyleByName method, StylesNum method, Style object

ReorderPageByID Method

ReorderPageByID Method

Places page into the specified position in the page collection of the document. The page to be repositioned is specified by the ID (the **ID** property) of the page in the page collection of the document. The position is indicated by the ID of the page, to which position the specified page must be placed.

Applies to: Document object

Syntax
[[**Let**] *booleanRet* =] *object*.**ReorderPageByID** (*pageIDFrom*, *pageIDTo*)

The **ReorderPageByID** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
pageIDFrom	Required. An expression that returns a **Long** value. Represents the ID (the **ID** property) of the page to be repositioned.
pageIDTo	Required. An expression that returns a **Long** value. Represents the ID (the **ID** property) of the page, to which position the page specified by *pageIDFrom* will be placed.
booleanRet	Optional. A **Boolean** type variable.

Remarks

If there is no page with the *pageIDFrom* or *pageIDTo* ID in the page collection of the document, the **ReorderPageByID** doesn't reorder the page and returns **False**. If the page was repositioned successfully, this method returns **True**.

See Also ID property, AddPage method, FindPage method, Page method, PageByID method, PagesNum method, RemovePage method, RemovePageByID method, ReorderPage method, Page object

ReorderPage Method

ReorderPage Method

Places page into the specified position in the page collection of the document. The page to be repositioned is specified by the index of the page in the page collection of the document. The position is indicated by the index, which the page will get after repositioning.

Applies to: Document object

Syntax

[[**Let**] *booleanRet* =] *object*.**ReorderPage** (*indexFrom*, *indexTo*)

The **ReorderPage** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
indexFrom	Required. An expression that returns a **Long** value. Represents the index of the page to be repositioned in the page collection of the document.

indexTo	Required. An expression that returns a **Long** value. Specifies the index of the position, to which the page will be placed.
booleanRet	Optional. A **Boolean** type variable.

Remarks

If either *indexFrom* or *indexTo* are less than **1** or greater than the number of pages in the document, the **RemoveMenuItem** method doesn't reposition the page and returns **False**. If the page was repositioned, the method returns **True**. Use the **PagesNum** method to find out the number of the pages in the document.

Note, that if *indexFrom* is greater than *indexTo*, then when you reposition a page with the *indexFrom* index, the indices of all pages, starting from *indexTo* to (*indexFrom* - 1), will be increased by **1**. If *indexTo* is greater than *indexFrom*, then when you reposition the page with the *indexFrom* index, the indices of all pages, starting from (*indexFrom* + 1) to *indexTo* will be decreased by **1**.

Example

This example contains a document-level script. The script uses the **ReorderPage** method to reverse the page order of the document.

```
' Declare variables
Dim page_count As Long
' Remember the number of the pages in the document
Let page_count = thisDoc.PagesNum()
' Reposition pages from the last position
' to the current, specified by the  i counter
For i=1 To page_count
    thisDoc.ReorderPage( page_count, i )
Next i
```

See Also AddPage method, FindPage method, Page method, PageByID method, PagesNum method, RemovePage method, RemovePageByID method, ReorderPageByID method, Page object

ReorderServObjByID Method

ReorderServObjByID Method

Places the service object into the specified position in the service object collection of the group/page. The repositioned service object is specified by its ID (the **ID** property). The position is specified by the ID of the service object, to whose position the repositioned service object will be placed.

Applies to: Page object, Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**ReorderServObjByID** (*servObjIDFrom*, *servObjIDTo*)

The **ReorderServObjByID** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
servObjIDFrom	Required. An expression that returns a **Long** value. The ID (**ID** property) of the service object to be repositioned.
servObjIDTo	Required. An expression that returns a **Long** value. The ID (**ID** property) of the service object to whose position the service object specified by *servObjIDFrom* will be placed.
booleanRet	Optional. A **Boolean** type variable.

Remarks

If there is no service object with the *servObjIDFrom* or *servObjIDTo* ID in the collection, the **ReorderServObjByID** method doesn't reposition the service object and returns **False**. If repositioning has been successful, the method returns **True**.

See Also	ID property, RemoveAllServObjs method, RemoveServObj method, RemoveServObjByID method, ReorderServObj method, ServObj method, ServObjByID method, ServObjsNum method, ServObj object

ReorderServObj Method

ReorderServObj Method

Places the service object into the specified position in the service object collection of the group/page. The repositioned service object is specified by its index in the service object collection of the page/group. The position is specified by the new index the service object will have after repositioning.

Applies to: Page object, Shape object

Syntax
[[**Let**] *booleanRet* =] *object*.**ReorderServObj** (*indexFrom*, *indexTo*)

The **ReorderServObj** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
indexFrom	Required. An expression that returns a **Long** value. The index of the service object to be repositioned in the service object collection of the group/page.
indexTo	Required. An expression that returns a **Long** value. The index of the position into which the service object will be placed.
booleanRet	Optional. A **Boolean** type variable.

Remarks

If either the *indexFrom* or *indexTo* are less than **1** or greater than the number of service objects in the group/page, the **ReorderServObj** method doesn't reposition the service object and returns **False**. If the service object has been repositioned successfully, the method returns **True**. Use the **ServObjsNum** method to find out the number of service objects in the group/page.

Note, that if *indexFrom* is greater than *indexTo*, then when a service object with the *indexFrom* index is repositioned, the indices of all service objects starting from *indexTo* and to (*indexFrom* - 1) will be increased by **1**. If *indexTo* is greater than *indexFrom*, then when a service object with the *indexFrom* index is repositioned, the indices of all service objects starting from (*indexFrom* + 1) and to *indexTo* will be decreased by **1**.

See Also	RemoveAllServObjs method, RemoveServObj method, RemoveServObjByID method, ReorderServObjByID method, ServObj method, ServObjByID method, ServObjsNum method, ServObj object

ReorderShapeByID Method

ReorderShapeByID Method

Places the shape into the specified position in the shape collection of the group/page. The repositioned shape is specified by its ID (the **ID** property). The position is specified by the ID of the shape, to whose position the repositioned shape will be placed.

Applies to: Page object, Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**ReorderShapeByID** (*shapeIDFrom*, *shapeIDTo*)

The **ReorderShapeByID** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
shapeIDFrom	Required. An expression that returns a **Long** value. The ID (**ID** property) of the service shape to be repositioned.
shapeIDTo	Required. An expression that returns a **Long** value. The ID (**ID** property) of the shape to whose position the shape specified by *shapeIDFrom* will be placed.
booleanRet	Optional. A **Boolean** type variable.

Remarks

If there is no shape with the *shapeIDFrom* or *shapeIDTo* ID in the collection, the **ReorderShapeByID** method doesn't reposition the shape and returns **False**. If repositioning has been successful, the method returns **True**.

See Also ID property, RemoveAllShapes method, RemoveShape method, RemoveShapeByID method, ReorderShape method, Shape method, ShapeByID method, ShapesNum method

ReorderShape Method

ReorderShape Method

Places the shape into the specified position in the shape collection of the group/page. The repositioned shape is specified by its index in the shape collection of the page/group. The position is specified by the new index the shape will get after repositioning.

Applies to: Page object, Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**ReorderShape** (*indexFrom*, *indexTo*)

The **ReorderShape** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
indexFrom	Required. An expression that returns a **Long** value. The index of the shape to be repositioned in the shape collection of the group/page.
indexTo	Required. An expression that returns a **Long** value. The index of the position into which the shape will be placed.
booleanRet	Optional. A **Boolean** type variable.

Remarks

If either the *indexFrom* or *indexTo* are less than **1** or greater than the number of shapes in the group/page, the **ReorderShape** method doesn't reposition the shape and returns **False**. If the shape has been repositioned successfully, the method returns **True**. Use the **ShapesNum** method to find out the number of shapes in the group/page.

Note, that if *indexFrom* is greater than *indexTo*, then when a shape with the *indexFrom* index is repositioned, the indices of all shapes starting from *indexTo* and to (*indexFrom* - 1) will be increased by **1**. If *indexTo* is greater than *indexFrom*, then when a shape with the *indexFrom* index is repositioned, the indices of all shapes starting from (*indexFrom* + 1) and to *indexTo* will be decreased by **1**.

See Also RemoveAllShapes method, RemoveShape method, RemoveShapeByID method, ReorderShapeByID method, Shape method, ShapeByID method, ShapesNum method

Restore Method

Restore Method

Restores the initial size and position of the window.

Applies to: Window object

Syntax

object.**Restore** ()

The **Restore** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns a **Window** object.

Remarks

The initial size and position of the window is its state when it's neither minimized, nor maximized. The initial size and position of the window can be set by using the **SetWindowRect** method. To find out the current state of the window, use the **State** property.

See Also	State property, Maximize method, Minimize method, SetWindowRect method

RowCount Method

RowCount Method

Returns the number of non-empty string, ie rows that contain data in a tabular representation of the CSV file data source.**Applies to:** DataSource object

Syntax

[[**Let**] *countRet* =] *object*.**RowCount** ()

The **RowCount** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the **DataSource** object.
countRet	Optional. A **Long** type variable.

Remarks

An instance of the DataSource object can be obtained using methods of the Shape.

Example

```
dim ds as DATASOURCE
dim count as Integer
ds = thisShape.DATASOURCE(1)
count = ds.RowCount()
```

```
trace count
```

See Also ColCount method

SaveAs Method (Document object)

SaveAs Method (Document object)

Saves the document with the specified parameters: filename, document format version, workspace, etc.

Applies to: Document object

Syntax

[[**Let**] *booleanRet* =] *object*.**SaveAs** (*fileName*, *saveFlags*, *saveVersion*, *showSaveDlg*)

The **SaveAs** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Document** object.
fileName	Required. An expression that returns a **String** value. The name of the file, to save the document in.
saveFlags	Required. An expression that returns a **Long** value. The flags describing the contents of the saved document.
saveVersion	Required. An expression that returns a **Long** value. The version of the format to save the document in.
showSaveDlg	Optional. An expression that returns a **Boolean** value. A flag that indicates whether to show the file save dialog: **True** - display the dialog, **False** - not to display the dialog and use the name, specified by the *fileName* parameter. The default value is **False**.
booleanRet	Optional. A **Boolean** type variable.

Remarks

The *saveFlags* parameter can take the following values:

Constant	**Value**	**Description**

cdSaveWithWS	&H1	Save the workspace information together with the document.
cdSaveAsTemplate	&H2	Save the document as template
cdSaveInXML	&H4	Save the document in XML format

The document format version *saveVersion* specifies in which format to save the document. The *saveVersion* parameter can take the following values: **200** or greater - the document is in the ConceptDraw V format, between **0** and **200** - the document is in the ConceptDraw 1.x format. To save the document in the same format it was saved before, set *saveVersion* to **0** or less.

If the document has been saved successfully with the specified parameters, the **SaveAs** method returns **True**. Otherwise, the method returns **False**.

Saving the document under *fileName* different from the current document filename changes the corresponding properties of the document: the **FullName** property, the **Name** property, the **Path** property. If the name of the file, specified in *fileName* equals to an empty string, the method attempts to save the document under the current filename. If the document hasn't been yet saved, the name of the file (the **Name** property) is made up automatically of the document title (the **Title** property) and the standard extension for ConceptDraw documents (**.cdd**) and then the document is saved in the current folder of the application. Also, by using the *showSaveDlg* parameter it's possible to specify the filename manually.

See Also OpenDoc method, Save method

SaveAs Method (Library object)

SaveAs Method (Library object)

Saves the library with the specified parameters: filename, version.

Applies to: Library object

Syntax

[[**Let**] *booleanRet* =] *object*.**SaveAs** (*fileName*, *saveVersion*, [*showSaveDlg*])

The **SaveAs** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the **Library** object.
fileName	Required. An expression that returns a **String** value. The filename (full or relative), under which the library is to be saved.
saveVersion	Required. An expression that returns a **Long** value. The version of the format to save the library in.
showSaveDlg	Optional. An expression that returns a **Boolean** value. A flag that indicates whether to show the file save dialog: **True** - display the dialog, **False** - not to display the dialog and use the name, specified by the *fileName* parameter. The default value is **False**.
booleanRet	Optional. A **Boolean** type variable.

Remarks

The document format version *saveVersion* specifies in which format to save the library. The *saveVersion* parameter can take the following values: **200** or greater - the library is in the ConceptDraw V format, between **0** and **200** - the library is in the ConceptDraw 1.x format. To save the library in the same format it was saved before, set *saveVersion* to **0** or less.

If the library has been saved successfully with the specified parameters, the **SaveAs** method returns **True**. Otherwise, the method returns **False**.

Saving the library under *fileName* different from the current document filename changes the corresponding properties of the library: the **FullName** property, the **Name** property, the **Path** property. If the name of the file, specified in *fileName* equals to an empty string, the method attempts to save the library under the current filename. If the library hasn't been yet saved, the name of the file (the **Name** property) is made up automatically of the library title (the **Title** property) and the standard extension for ConceptDraw libraries (**.cdl**) and then the library is saved in the current folder of the application. Also, by using the *showSaveDlg* parameter it's possible to specify the filename manually.

See Also FullName property, Name property, Path property, OpenLib method, Save method

SaveWorkspace Method

SaveWorkspace Method

Saves the workspace with the specified parameters: filename and workspace format version.

Applies to: Application object

Syntax

[[**Let**] *booleanRet* =] *object*.**SaveWorkspace** (*fileName*, *saveVersion*, *showSaveDlg*)

The **SaveAs** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the **Application** object.
fileName	Required. An expression that returns a **String** value. The name of the file, to save the document in.
saveVersion	Required. An expression that returns a **Long** value. The version of the format to save the workspace in.
showSaveDlg	Optional. An expression that returns a **Boolean** value. A flag that indicates whether to show the file save dialog: **True** - display the dialog, **False** - not to display the dialog and use the name, specified by the *fileName* parameter. The default value is **False**.
booleanRet	Optional. A **Boolean** type variable.

Remarks

The workspace format version *saveVersion* specifies in which format to save the workspace. The *saveVersion* parameter can take the following values: **200** or greater - the workspace is in the ConceptDraw V format, between **0** and **200** - the workspace is in the ConceptDraw 1.x format.

If the workspace has been saved successfully with the specified parameters, the **SaveWorkspace** method returns **True**. Otherwise, the method returns **False**.

Save Method (Document object)

Save Method (Document object)

Saves the document.

Applies to: Document object

Syntax

[[**Let**] *booleanRet* =] *object*.**Save** (*saveFlags*)

The **Save** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Document** object.
saveFlags	Optional. An expression that returns a **Long** value. Flags of the document saving options.
booleanRet	Optional. A **Boolean** type variable.

Remarks

The method attempts to save the document under the default filename (the **FullName** property) and in the format in which it was saved earlier. If the document hasn't been yet saved, the name of the file (the **Name** property) is made up automatically of the document title (the **Title** property) and the standard extension for ConceptDraw documents (**.cdd** for a regular document, and **.cdx** for documents in the ConceptDraw XML format), and then the document is saved in the current folder of the application.

The option flags *saveFlags* can take the following value:

Constant	Value	Description
cdSaveWithWS	&H1	Save the workspace information together with the document.

If the document has been saved successfully, the **Save** method returns **True**. Otherwise, the method returns **False**. To save the document with the specified filename and parameters, use the **SaveAs** method.

See Also FullName property, Name property, Title property, OpenDoc method, SaveAs method

Save Method (Library object)

Save Method (Library object)

Saves the library.

Applies to: Library object

Syntax

[[**Let**] *booleanRet* =] *object*.**Save** ()

The **Save** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Library** object.
booleanRet	Optional. A **Boolean** type variable.

Remarks

The method attempts to save the library under the default filename (the **FullName** property). If the library hasn't been yet saved, the name of the file (the **Name** property) is made up automatically of the library title (the **Title** property) and the standard extension for ConceptDraw libraries (**.cdl**) and then the library is saved in the current folder of the application.

If the library has been saved successfully, the **Save** method returns **True**. Otherwise, the method returns **False**. To save the library with the specified filename and parameters, use the **SaveAs** method.

See Also FullName property, Name property, Title property, OpenLib method, SaveAs method

ScrollViewTo Method

ScrollViewTo Method

Scrolls a window to a Elementicular page coordinate.

Applies to: Window object

Syntax

object.**ScrollViewTo** (*x*, *y*)

The **ScrollViewTo** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Window** object.
x	Required. An expression that returns a **Double** value. The x-coordinate to which to scroll.

y	Required. An expression that returns a **Double** value. The y-coordinate to which to scroll.

Remarks

This method is only effective if the window is of the document view type (see the **Type** property). For windows of all other types, the **ScrollViewTo** method always returns **0**.

The method scrolls the window so that the point with the *x* and *y* coordinates is displayed in the center of the window. The coordinates are specified in the coordinate system of the page or the shape, displayed in the window. The units of measure are internal units (**InternalUnit**). Use the **ViewCenterX** and **ViewCenterY** properties to get the coordinates of the point of the page or group displayed in the center of the window.

See Also	Type property, ViewCenterX property, ViewCenterY property, ViewZoom property

SegmentsNum Method

SegmentsNum Method

Returns the number of segments in geometry.**Applies to:** Geomentry object

Syntax
object.**SegmentNum** ()

The **SegmentNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the **Geomentry** object.

Remarks

Knowing the number of segments in geometry, you can get any information about it. To get the values and formulas specific fields, use the methods of the Shape of Get ... Property (), and GetPropertyFormula ().

Example

```
dim gm as Geometry
dim count as Integer
gm = thisShape.Geometry(1)
count = gm.SegmentsNum()
trace count
```

See Also ColorProperty method, Geometry method, GetBooleanProperty method, GetByteProperty method, GetDoubleProperty method, GetIntegerProperty method, GetLongProperty method, GetPropertyFormula method, GetSingleProperty method, GetStringProperty method, Geometry object, Shape object

SelectAll Method

SelectAll Method

Selects all shapes of the page or group displayed in the window.

Applies to: Window object

Syntax

[[**Let**] *boolRet* =] *object*.**SelectAll** ()

The **SelectAll** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Window** object.
boolRet	Optional. A **Boolean** type variable.

Remarks

This method is only effective if the window is of the document view type (see the **Type** property). For windows of all other types, the **SelectAll** method always returns **False**.

The **SelectAll** method selects all shapes of the page or group displayed in the window, and returns **True** if at least one shape has been selected in the result. If all shapes of the page or group displayed in the window are already selected, returns **False**.The inverse method to **SelectAll** is the **DeselectAll** method, which removes selection from all the shape of the page or group.

See Also	Type property, Deselect method, DeselectAll method, GetSelectedService method, GetSelectedShape method, Select method, SelectedNum method

SelectedNum Method

SelectedNum Method

Returns the number of selected shapes on the page/group being displayed.

Applies to: Window object

Syntax

[[**Let**] *selectedNumRet* =] *object*.**SelectedNum** ()

The **SelectedNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Window** object.
selectedNumRet	Optional. A **Long** type variable.

Remarks

This method is only effective if the window is of the document view type (see the **Type** property). For windows of all other types the **SelectedNum** method always returns **0**.

See Also	Type property, Deselect method, DeselectAll method, GetSelectedService method, GetSelectedShape method, Select method, SelectAll method

Select Method

Select Method

Selects a shape with the specified ID (the **ID** property).

Applies to: Window object

Syntax

[[**Let**] *boolRet* =] *object*.**Select** (*shapeID*)

The **Select** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Window** object.
shapeID	Required. An expression that returns a **Long** value. Specifies the ID of the shape.
boolRet	Optional. A **Boolean** type variable.

Remarks

This method is only effective if the window is of the document view type (see the **Type** property). For windows of all other types, the **Select** method always returns **False**.

If there is a shape with the specified **ID** in the shape collection of the page or group, displayed in the window, the **Select** method selects the shape and returns **True**. Otherwise (or if the object is already selected), the **Select** method returns **False**. This method doesn't deselect already selected shapes on the page or in the group. The inverse method to **Select** is the **Deselect** method, which removes selection from the shape with the specified ID (the **ID** property).

See Also	ID property, Type property, Deselect method, DeselectAll method, GetSelectedService method, GetSelectedShape method, SelectAll method, SelectedNum method

SendBack Method

SendBack Method

Moves the object (shape) in the first position in the collection of objects (shapes) of the parent group. Returns the index of the object (shape) in a collection of objects (shapes) of the parent group.

Applies to: Shape object

Syntax
[[**Let**] *index* =] *object*.**SendBack** ()

The **SendBack** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
index	Optional. A **Long** type variable.

Remarks

If the object is an object directly on the document page, then the parent of this object is a page (Property Page). If an object is placed in a group, then its parent is a group of objects. The numbering of objects starting with 0. In case of success the function returns 0 (first position in the collection of objects (shapes) of the parent group). In case of error the method returns -1.

Example
```
dim index as Integer
index = thisShape.SendBack()
trace index
```

See Also GetIndex method, Page property, Parent property, SendFront method, StepBack method, StepFront method

SendFront Method

SendFront Method

Moves the object (shape) in the last position in the collection of objects (shapes) of the parent group. Returns the index of the object (shape) in a collection of objects (shapes) of the parent group.

Applies to: Shape object

Syntax
[[Let] *index* =] *object*. **SendFront** ()

The **SendFront** method syntax has these Elements:

Element	**Description**

object	Required. An expression that returns a **Shape** object.
index	Optional. A **Long** type variable.

Remarks

If the *object* is an object directly on the document page, then the parent of this object is a page (Property **Page).** If an object is placed in a group, then its parent is a group of objects. The numbering of objects starting with 0. In case of error the method returns -1.

Example

```
dim index as Integer
index = thisShape.SendFront()
trace index
```

See Also GetIndex method, Page property, Parent property, SendBack method, StepBack method, StepFront method

ServObjByID Method

ServObjByID Method

Searches for a service object with the specified ID (the **ID** property) in the service object collection of the group/page. Returns a **ServObj** object that corresponds to the found service object.

Applies to: Page object, Shape object

Syntax

[[**Set**] *servObjRet* =] *object*.**ServObjByID** (*servObjID*)

The **ServObjByID** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
servObjID	Required. An expression that returns a **Long** value. The ID of the service object.
servObjRet	Optional. A **ServObj** type variable.

Remarks

If there is no service object with the *servObjID* ID in the service object collection of the group/page, the **ServObjByID** method returns **Nothing**.

See Also	ID property, RemoveAllServObjs method, RemoveServObj method, RemoveServObjByID method, ReorderServObj method, ReorderServObjByID method, ServObj method, ServObjsNum method, ServObj object

ServObjsNum Method

ServObjsNum Method

Returns the number of service objects in a group/page.

Applies to: Page object, Shape object

Syntax

[[**Let**] *countRet* =] *object*.**ServObjsNum** ()

The **ServObjsNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
countRet	Optional. A **Long** type variable.

Remarks

If there are no service objects in the group/page, the **ServObjsNum** returns **0**.

See Also	RemoveAllServObjs method, RemoveServObj method, RemoveServObjByID method, ReorderServObj method, ReorderServObjByID method, ServObj method, ServObjByID method, ServObj object

ServObj Method

ServObj Method

Returns a **ServObj** object that corresponds to the service object with the specified index in the service object collection of the page/group.

Applies to: Page object, Shape object

Syntax

[[**Set**] *servObjRet* =] *object*.**ServObj** (*index*)

The **ServObj** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
index	Required. An expression that returns a **Long** value.
servObjRet	Optional. A **ServObj** type variable.

Remarks

If *index* is less than **1** or greater than the number of service objects in a group or page to which *object* corresponds, the **ServObj** method returns **Nothing**. You can use the **ServObjsNum** method to find out the number of service objects in the group or page.

See Also	RemoveAllServObjs method, RemoveServObj method, RemoveServObjByID method, ReorderServObj method, ReorderServObjByID method, ServObjByID method, ServObjsNum method, ServObj object

SetActiveLib Method

SetActiveLib Method

Makes the specified library active.

Applies to: Application object

Syntax
[[**Let**] *booleanRet* =] *object*.**SetActiveLib** (*libraryObj*)

The **SetActiveLib** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns an instance of the **Application** object.
libraryObj	Required. An expression, that returns an instance of the **Library** object. Indicates the library to make active.
booleanRet	Optional. A **Boolean** type variable.

Remarks

If *libraryObj* is open in the application, the **SetActiveLib** changes the active library from the current to specified one, and returns **True**. Otherwise the **SetActiveLib** method remains the active library unchanged and returns **False**. The **SetActiveLib** method is used to change the **ActiveLib** property.

Example

This example contains a application-level script. It activates the fifth library from the library collection of the application.

```
thisApp.SetActiveLib( thisApp.Lib(5) )
```

See Also ActiveLib property, Library object

SetActivePageByID Method

SetActivePageByID Method

Makes active the page of the document with the specified ID (the **ID** property).

Applies to: Document object

Syntax
object.**SetActivePageByID** (*pageID*)

The **SetActivePageByID** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
pageID	Required. An expression that returns a **Long** value. The ID (**ID** property) of the page to make active.

Remarks

If there is no page with the specified *pageID* in the document, the **SetActivePageByID** method doesn't change the active page of the document. To get a **Page** object, that corresponds to the active page of the document, use the **ActivePage** property.

See Also ActivePage property, SetActivePage method, Page object

SetActivePage Method

SetActivePage Method

Makes active the page with the specified index in the page collection of the document.

Applies to: Document object

Syntax

object.**SetActivePage** (*index*)

The **SetActivePage** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
index	Required. An expression that returns a **Long** value. The index of the page in the page collection of the document.

Remarks

If *index* is less than **1** or greater than the number of pages in the page collection of the document, the **SetActivePage** method doesn't change the active page of the document. To get a reference to the instance of the **Page** object, that corresponds to the active page of the document, use the **ActivePage** property.To find out the number of pages in the document, use the **PagesNum** method.

See Also ActivePage property, PagesNum method, SetActivePageByID method, Page object

SetActiveView Method

SetActiveView Method

Activates the specified window of the document.

Applies to: Document object

Syntax

[[**Let**] *booleanRet* =] *object*.**SetAcitveView** (*viewID*)

The **SetActiveView** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
viewID	Required. An expression that returns a **Long** value. The ID (**ID** property) of the window of the document to be activated.
booleanRet	Optional. A **Boolean** type variable.

Remarks

Note, that activating the document window directly modifies the value of the **ActiveView** property. If there is no window with the specified *viewID* in the window collection of the document, the **SetActiveView** method doesn't change the current active document view.

See Also ActiveView property, Window object

SetBooleanProperty Method

SetBooleanProperty Method

Sets the value of a **Boolean** type property.

Applies to objects: Shape

Syntax

object.**SetBooleanProperty**(*data*, *propTag* [, *num*[, *geom*]])

The **SetBooleanProperty** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the Shape object.
data	Required. An expression that returns a **Boolean** value. The value to be set to the property.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method,

SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

SetByteProperty Method

SetByteProperty Method

Sets the value of a **Byte** type property.

Applies to: Shape object

Syntax

object.**SetByteProperty**(*data*, *propTag* [, *num*[, *geom*]])

The **SetByteProperty** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the Shape object.
data	Required. An expression that returns a **Byte** value. The value to be set to the property.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

SetCharColor Method

SetCharColor Method

Sets the color for the specified character block of the shape's text.

Applies to: Shape object

Syntax

object.**SetCharColor** (*iFrom*, *iTo*, [*irc*], [*gm*], [*by*], [*bk*], [*bTransparent*])

The **SetCharColor** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
iFrom	Required. An expression that returns a **Long** value. The index of the first character in the character block.
iTo	Required. An expression that returns a **Long** value. The index of the last character in the character block.
irc	Optional. An expression that returns an **Integer** value. The possible options for this parameter: - Index color: the index of the color in the color palette of the document which owns the *object* shape. The valid range is from **1** to the number of the colors in the color palette of the document. - RGB color: the red component of the color. The valid range is from **0** to **255**.

	- CMYK color: the **cyan** component of the color. The valid range is from **0** to **100**. The default value is **-1**.
gm	Optional. An expression that returns an **Integer** value. The possible options for this parameter: - RGB color: the green component of the color. The valid range is from **0** to **255**. - CMYK color: the **magenta** component of the color. The valid range is from **0** to **100**. The default value is **-1**.
by	Optional. An expression that returns an **Integer** value. The possible options for this parameter: - RGB color: the blue component of the color. The valid range is from **0** to **255**. - CMYK color: the **yellow** component of the color. The valid range is from **0** to **100**. The default value is **-1**.
bk	Optional. An expression that returns an **Integer** value. The possible options for this parameter: - CMYK color: the **black** component of the color. The valid range is from **0** to **100**. The default value is **-1**.
bTransparent	Optional. An expression that returns a **Boolean** value. A flag which is **True** when the character block is transparent, and **False** if the character block is not transparent. The default value is **False**.

Remarks

Note, that if the character block doesn't include the character block within the *iFrom* to *iTo* range, the **SetCharColor** method adds a new character block and sets the specified color to it.

The format of the color for the character block is described in the following way. If all four components of a CMYK color are set, that is, the *irc*, *gm*, *by* and *bk* parameters are equal to or greater than **0** and less than or equal to **100**, the color is considered a CMYK color. Otherwise, if only the RGB components are set, that is, the *irc*, *gm*, *by* parameters are equal to or greater than **0** and less than or equal to **255**, and *bk* is less than **0** or greater than **100**, the color is considered an RGB color. Otherwise, the color is considered an indexed color with the *irc* index in the color palette of the document - if the *irc* parameter is greater than or equal to **1** and less than or equal to the number of the colors in the color palette of the document which owns the *object* shape. In all other cases the color of the character block is not altered and only the transparency parameter *bTransparent* is applied.

See Also	Character method, CharactersNum method, GetCharacterIndex method, RemoveCharacter method, SetCharFont method, SetCharHyperlink method, SetCharLanguage method, SetCharPos method, SetCharSize method, SetCharSpacing method, SetCharStyle method, Character object

SetCharFont Method

SetCharFont Method

Sets the font for the specified character block of the shape's text.

Applies to: Shape object

Syntax

object.**SetCharFont** (*iFrom*, *iTo*, *iFont*)

The **SetCharFont** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character in the character block.
iTo	Required. An expression that returns a **Long** value. The index of the last character in the character block.
iFont	Required. An expression that returns a **Long** value. A font index in the font collection of the document which owns the *object* shape.

Remarks

Note, that if the character block doesn't include the character block within the *iFrom* to *iTo* range, the **SetCharFont** method adds a new character block and sets the specified font to it.

The value of *iFont* must be greater than or equal to **1** and less than or equal to the number of the fonts in the font collection of the document. To find out the number of fonts in the document, use the **FontsNum** method.

See Also	Character method, CharactersNum method, GetCharacterIndex method, RemoveCharacter method, SetCharColor method, SetCharHyperlink method,

SetCharLanguage method, SetCharPos method, SetCharSize method, SetCharSpacing method, SetCharStyle method, Character object

SetCharHyperlink Method

SetCharHyperlink Method

Assigns a hyperlink to the specified character block of the shape's text.

Applies to: Shape object

Syntax

object.**SetCharHyperlink** (*iFrom, iTo, hyperlinkID*)

The **SetCharHyperlink** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character in the character block.
iTo	Required. An expression that returns a **Long** value. The index of the last character in the character block.
hyperlinkID	Required. An expression that returns a **Long** value. The ID (**ID** property) of the hyperlink to be assigned to the specified character block.

Remarks

Note, that if the character block doesn't include the character block within the *iFrom* to *iTo* range, the **SetCharHyperlink** method adds a new character block and assigns the specified hyperlink to it.

If there is no hyperlink with the *hyperlinkID* ID in the hyperlink collection of the document which owns the *object* shape, the **SetCharHyperlink** neither adds a new character block, nor assigns a hyperlink to an existing one.

See Also Hyperlink property, ID property, Character method, CharactersNum method, GetCharacterIndex method, RemoveCharacter method, SetCharColor method, SetCharFont method, SetCharLanguage method, SetCharPos method, SetCharSize method, SetCharSpacing method, SetCharStyle method, Character object,

SetCharLanguage Method

SetCharLanguage Method

Assigns a charset for the specified character block of the shape's text.

Applies to: Shape object

Syntax

object.**SetCharLanguage** (*iFrom*, *iTo*, *Language*)

The **SetCharLanguage** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character in the character block.
iTo	Required. An expression that returns a **Long** value. The index of the last character in the character block.
Language	Required. An expression that returns a **Byte** value. The charset.

Remarks

Note, that if the character block collection of the shape doesn't include the character block within the *iFrom* to *iTo* range, the **SetCharLanguage** method adds a new character block and sets the specified charset to it.

Below is the list of possible values of the *Language* parameter:

Constant	Value	Description
ANSI_CHARSET	0	ANSI charset.
DEFAULT_CHARSET	1	Default charset.
SYMBOL_CHARSET	2	Symbol charset.
MAC_CHARSET	77	Macintosh charset.
SHIFTJIS_CHARSET	128	charset.
HANGEUL_CHARSET	129	Hungarian charset.

HANGUL_CHARSET	129	Hungarian charset.
JOHAB_CHARSET	130	charset.
GB2312_CHARSET	134	charset.
CHINESEBIG5_CHARSET	136	Chinese charset.
GREEK_CHARSET	161	Greek charset.
TURKISH_CHARSET	162	Turkish charset.
VIETNAMESE_CHARSET	163	Vietnamese charset.
HEBREW_CHARSET	177	Hebrew charset.
ARABIC_CHARSET	178	Arabic charset.
BALTIC_CHARSET	186	Baltic charset.
RUSSIAN_CHARSET	204	Russian (cyrillic) charset.
THAI_CHARSET	222	Thai charset.
EASTEUROPE_CHARSET	238	East Europe charset.
OEM_CHARSET	255	OEM charset.

See Also	Language property, Character method, CharactersNum method, GetCharacterIndex method, RemoveCharacter method, SetCharColor method, SetCharFont method, SetCharHyperlink method, SetCharPos method, SetCharSize method, SetCharSpacing method, SetCharStyle method, Character object

SetCharPos Method

SetCharPos Method

Sets the position of the character block with respect to the baseline of the shape's text.

Applies to: Shape object

Syntax

object.**SetCharPos** (*iFrom*, *iTo*, *Pos*)

The **SetCharPos** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character in the character block.
iTo	Required. An expression that returns a **Long** value. The index of the last character in the character block.
Pos	Required. An expression that returns a **Byte** value. The text position.

Remarks

Note, that if the character block collection of the shape doesn't include the character block within the *iFrom* to *iTo* range, the **SetCharPos** method adds a new character block and sets the specified position to it.

Below is the list of possible values of the *Pos* parameter:

Constant	Value	Description
cdPosNormal	0	Normal size and position of the text.
cdPosSuper	1	Superscript.
cdPosSub	2	Subscript.

See Also	Pos property (Character object), Character method, CharactersNum method, GetCharacterIndex method, RemoveCharacter method, SetCharColor method, SetCharFont method, SetCharHyperlink method, SetCharLanguage method, SetCharSize method, SetCharSpacing method, SetCharStyle method, Character object

SetCharSize Method

SetCharSize Method

Sets the font size for the specified character block of the shape's text.

Applies to: Shape object

Syntax

object.**SetCharSize** (*iFrom*, *iTo*, *fontSize*)

The **SetCharSize** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character in the character block.
iTo	Required. An expression that returns a **Long** value. The index of the last character in the character block.
fontSize	Required. An expression that returns an **Integer** value. The font size.

Remarks

The font size (*fontSize* parameter) is specified in **points** (1 pt = 1/72 inch).

Note, that if the character block collection of the shape doesn't include the character block within the *iFrom* to *iTo* range, the **SetCharSize** method adds a new character block and sets the specified font size to it.

See Also	Character method, CharactersNum method, GetCharacterIndex method, RemoveCharacter method, SetCharColor method, SetCharFont method, SetCharHyperlink method, SetCharLanguage method, SetCharPos method, SetCharSize method, SetCharSpacing method, SetCharStyle method, Character object

SetCharSpacing Method

SetCharSpacing Method

Sets the character spacing for the specified character block of the shape's text.

Applies to: Shape object

Syntax

object.**SetCharSpacing** (*iFrom*, *iTo*, *charSpacing*)

The **SetCharSpacing** method syntax has these Elements:

Element	Description

object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character in the character block.
iTo	Required. An expression that returns a **Long** value. The index of the last character in the character block.
charSpacing	Required. An expression that returns a **Single** value. The character spacing.

Remarks

Note, that if the character block collection of the shape doesn't include the character block within the *iFrom* to *iTo* range, the **SetCharSpacing** method adds a new character block and applies the specified character spacing to it.

See Also	Character method, CharactersNum method, GetCharacterIndex method, RemoveCharacter method, SetCharColor method, SetCharFont method, SetCharHyperlink method, SetCharLanguage method, SetCharPos method, SetCharSize method, SetCharStyle method, Character object

SetCharStyle Method

SetCharStyle Method

Sets the font style (bold, italic, underline, etc.) for the specified character block of the shape's text.

Applies to: Shape object

Syntax

object.**SetCharStyle** (*iFrom*, *iTo*, *bStyle*)

The **SetCharStyle** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character in the character block.
iTo	Required. An expression that returns a **Long** value. The index of the last character in the character block.

bStyle	Required. An expression that returns a **Byte** value. The font style.

Remarks

Note, that if the character block collection of the shape doesn't include the character block within the *iFrom* to *iTo* range, the **SetCharStyle** method adds a new character block and sets the specified font style to it.

Below is the list of possible values of the *bStyle* parameter:

Constant	**Value**	**Description**
cdFSNormal	0	Normal.
cdFSBold	1	Bold.
cdFSItalic	2	Italic.
cdFSUnderline	4	Underline.
cdFSStrikeTrough	8	Strikethrough.

See Also Character method, CharactersNum method, GetCharacterIndex method, RemoveCharacter method, SetCharColor method, SetCharFont method, SetCharHyperlink method, SetCharLanguage method, SetCharPos method, SetCharSize method, SetCharSpacing method, Character object

SetCmdProcessing Method

SetCmdProcessing Method

Sets a procedure to process the menu item command.

Applies to: MenuItem object

Syntax

[[**Set**] *bRet* =] *object*.**SetCmdProcessing** (*sOnCmdSub* [, *sOnCmdModule*])

The **MenuItemByCmdID** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **MenuItem** object.

sOnCmdSub	Required. An expression that returns a **String** value. Represents the processing procedure name.
sOnCmdModule	Optional. An expression that returns a **String** value. Represent the external module (shared library on the Mac or *.dll on the Windows platform).
bRet	Optional. A **Boolean** type variable.

Remarks

The **SetCmdProcessing** method returns **True** if setting was successfull and **False** in other case.

SetCMYK Method

SetCMYK Method

Sets color components in CMYK format.

Applies to: Color object, ColorEntry object

Syntax

object.**SetCMYK** (*cyan*, *magenta*, *yellow*, *black*)

The **SetCMYK** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object in the **Applies to** list.
cyan	Required. An expression that returns an **Integer** value (from 0 to 100).
magenta	Required. An expression that returns an **Integer** value (from 0 to 100).
yellow	Required. An expression that returns an **Integer** value (from 0 to 100).
black	Required. An expression that returns an **Integer** value (from 0 to 100).

Remarks

After the **SetCMYK** method has been called, the color is converted to the CMYK format regardless of its previous format.

Example

This example demonstrates how to change the fill color of a rectanlge in CMYK format.

```
dim s as shape
```

```
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
s.FillColor.SetCMYK(33,25,66,5)  ' Change the Shape's fill color in CMYK
format
s.PropertyChanged(CDPT_FILLCOLOR)
```

See Also SetRGB Method

SetDefaultFormula Method

SetDefaultFormula Method

Creates a default formula for the property.

Applies to: Shape object, ServObj

Syntax

object.**SetDefaultFormula**(*propTag* [, *num*[, *geom*]])

The **SetDefaultFormula** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. Only for Shape object. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object and **ServObj** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

SetDoubleProperty Method

SetDoubleProperty Method

Sets the value of a **Double** type property.

Applies to objects: Shape, ServObj

Syntax

object.**SetDoubleProperty**(*data*, *propTag* [, *num*[, *geom*]])

The **SetDoubleProperty** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
data	Required. An expression that returns a **Double** value. The value to be set to the property.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. It is used only for the objectShape. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.

geom	Optional. It is used only for the objectShape. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object and **ServObj** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag, num, geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

SetFillColor Method

SetFillColor Method

Sets the fill color (pattern) of an object (shape) for the current style of the document.

Applies to: Style object

Syntax

object. **SetFillColor** *(r, g, b)*

The **SetFillColor** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Style** object.

r	Required. An expression that returns a **Byte** value. The red color component in RGB.Valid values range from 0 to 255.
g	Required. An expression that returns a **Byte** value. The green color component in RGB.Valid values range from 0 to 255.
b	Required. An expression that returns a **Byte** value. The blue color component in RGB.Valid values range from 0 to 255.

Remarks

For the fill color you can use the object **FillColor Style,** which is also a table setting **CDPT_STYLED_FILLBACKGND** object.

See Also Style object, FillColor Property, Property FillPatColor, penColor Property, PropertyShadowColor, Shadow
SetFillPatColor , setPenColorMethod, Method SetShadowColor, SetShadowPatColor Method.

SetFillPatColor Method

SetFillPatColor Method

Sets the color of the fill pattern of the object (shape) for the current style of the document.

Applies to: Style object

Syntax

object. **SetFillPatColor** *(r, g, b)*

The **SetFillPatColor** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Style** object.
r	Required. An expression that returns a **Byte** value. The red color component in RGB.Valid values range from 0 to 255.
g	Required. An expression that returns a **Byte** value. The green color component in RGB.Valid values range from 0 to 255.
b	Required. An expression that returns a **Byte** value. The blue color component in RGB.Valid values range from 0 to 255.

Remarks

For color fill pattern you can use the object **FillPatColor Style,** which is also a table setting **CDPT_STYLED_FILLFOREGND** object.

See Also	Style object, FillColor property, FillPatColor property, PenColor property, ShadowColor property, ShadowPatColor property, SetFillColor method, SetPenColor method, SetShadowColor method, SetShadowPatColor method.

SetIcon Method

SetIcon Method

Sets the image from a graphic file as the icon for the specified library shape.

Applies to: Master object

Syntax

[[**Let**] *booleanRet* =] *object*.**SetIcon** (*iconName*)

The **SetIcon** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Master** object.
iconName	Required. An expression that returns a **String** value. The name and path (full or relative) to the file that contains the icon image.
booleanRet	Optional. A **Boolean** type variable.

Remarks

If the file with the *iconName* name can't be opened, is not a graphic file, or is not supported by ConceptDraw, the **SetIcon** method doesn't change the current icon of the library shape and returns **False**. If the icon has been replaced successfully, the method returns **True**.

See Also	Equal method, SetShape method

SetIntegerProperty Method

Sets the value of an **Integer** type property.

Applies to objects: Shape

Syntax

object.**SetIntegerProperty**(*data*, *propTag* [, *num*[, *geom*]])

The **SetIntegerProperty** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the Shape object.
data	Required. An expression that returns a **Integer** value. The value to be set to the property.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag, num, geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method,

SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

SetLongProperty Method

SetLongProperty Method

Sets the value of a **Long** type property.

Applies to: Shape object

Syntax

object.**SetLongProperty**(*data*, *propTag* [, *num*[, *geom*]])

The **SetLongProperty** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of the Shape object.
data	Required. An expression that returns a **Long** value. The value to be set to the property.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag, num, geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

SetNullFormula Method

SetNullFormula Method

Deletes the formula of the specified table property of the shape.

Applies to: Shape object, ServObj

Syntax

object.**SetNullFormula**(*propTag* [, *num*[, *geom*]])

The **SetNullFormula** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object from the **Applies to** list.
propTag	Required. It is used only for the objectShape. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. It is used only for the objectShape. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object and **ServObj** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

SetParaAfterSpacing Method

SetParaAfterSpacing Method

Sets the distance between the specified paragraph and the next paragraph of the shape's text.

Applies to: Shape object

Syntax

object.**SetParaAfterSpacing** (*iFrom*, *iTo*, *AfterSpacing*)

The **SetParaAfterSpacing** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character of the paragraph.
iTo	Required. An expression that returns a **Long** value. The index of the last character of the paragraph.

AfterSpacing	Required. An expression that returns a **Single** value. The interval between this paragraph and the next one.

Remarks

Note, that if the paragraph collection of the shape doesn't contain the paragraph that corresponds to the character sequence within the *iFrom* to *iTo* range, the **SetParaAfterSpacing** method adds a new paragraph and applies the specified interval to it.

If the *iFrom* and *iTo* parameters were specified incorrectly, no changes are made. The paragraph spacing is measured in internal units (**InternalUnit**).

See Also	GetParagraphIndex method, Paragraph method, ParagraphsNum method, RemoveParagraph method, SetParaBeforeSpacing method, SetParaFirstInd method, SetParaHAlign method, SetParaLeftInd method, SetParaLineSpacing method, SetParaRightInd method, Paragraph object

SetParaBeforeSpacing Method

SetParaBeforeSpacing Method

Sets the distance between the specified paragraph and the previous paragraph of the shape's text.

Applies to: Shape object

Syntax

object.**SetParaBeforeSpacing** (*iFrom*, *iTo*, *BeforeSpacing*)

The **SetParaBeforeSpacing** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character of the paragraph.
iTo	Required. An expression that returns a **Long** value. The index of the last character of the paragraph.
BeforeSpacing	Required. An expression that returns a **Single** value. The interval between this paragraph and the previous one.

Remarks

Note, that if the paragraph collection of the shape doesn't contain the paragraph that corresponds to the character sequence within the *iFrom* to *iTo* range, the **SetParaBeforeSpacing** method adds a new paragraph and applies the specified interval to it.

If the *iFrom* and *iTo* parameters were specified incorrectly, no changes are made. The paragraph spacing is measured in internal units (**InternalUnit**).

See Also	GetParagraphIndex method, Paragraph method, ParagraphsNum method, RemoveParagraph method, SetParaAfterSpacing method, SetParaFirstInd method, SetParaHAlign method, SetParaLeftInd method, SetParaLineSpacing method, SetParaRightInd method, Paragraph object

SetParaFirstInd Method

SetParaFirstInd Method

Sets the first line indent for the specified paragraph of the shape.

Applies to: Shape object

Syntax

object.**SetParaFirstInd** (*iFrom*, *iTo, FirstInd*)

The **SetParaFirstInd** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character of the paragraph.
iTo	Required. An expression that returns a **Long** value. The index of the last character of the paragraph.
FirstInd	Required. An expression that returns a **Single** value. The first line indent value for the paragraph.

Remarks

Note, that if the paragraph collection of the shape doesn't contain the paragraph that corresponds to the character sequence within the *iFrom* to *iTo* range, the **SetParaFirstInd** method adds a new paragraph and applies the specified first line indent to it.

If the *iFrom* and *iTo* parameters were specified incorrectly, no changes are made. The first line indent size is measured in internal units (**InternalUnit**).

See Also	GetParagraphIndex method, Paragraph method, ParagraphsNum method, RemoveParagraph method, SetParaAfterSpacing method, SetParaBeforeSpacing method, SetParaHAlign method, SetParaLeftInd method, SetParaLineSpacing method, SetParaRightInd method, Paragraph object

SetParaHAlign Method

SetParaHAlign Method

Sets the horizontal alignment type of text with respect to its text box for the specified paragraph of the shape.

Applies to: Shape object

Syntax

object.**SetParaHAlign** (*iFrom*, *iTo*, *HAlign*)

The **SetParaHAlign** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character of the paragraph.
iTo	Required. An expression that returns a **Long** value. The index of the last character of the paragraph.
HAlign	Required. An expression that returns a **Single** value. The horizontal alignment type.

Remarks

Note, that if the paragraph collection of the shape doesn't contain the paragraph that corresponds to the character sequence within the *iFrom* to *iTo* range, the **SetParaHAlign** method adds a new paragraph and applies the specified horizontal alignment type to it.

The *HAlign* parameter can take the following values:

Constant	**Value**	**Description**
cdHorizLeft	0	Align to the left edge.
cdHorizCenter	1	Align to the center.
cdHorizRight	2	Align to the right edge.

See Also GetParagraphIndex method, Paragraph method, ParagraphsNum method, RemoveParagraph method, SetParaAfterSpacing method, SetParaBeforeSpacing method, SetParaFirstInd method, SetParaLeftInd method, SetParaLineSpacing method, SetParaRightInd method, Paragraph object

SetParaLeftInd Method

SetParaLeftInd Method

Sets the distance the paragraph's text is indented from the left edge of the text block.

Applies to: Shape object

Syntax

object.**SetParaLeftInd** (*iFrom*, *iTo*, *LeftInd*)

The **SetParaLeftInd** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character of the paragraph.
iTo	Required. An expression that returns a **Long** value. The index of the last character of the paragraph.
LeftInd	Required. An expression that returns a **Single** value. The paragraph indent from the left edge of the text box.

Remarks

Note, that if the paragraph collection of the shape doesn't contain the paragraph that corresponds to the character sequence within the *iFrom* to *iTo* range, the **SetParaLeftInd** method adds a new paragraph and applies the specified left indent to it.

If the *iFrom* and *iTo* parameters were specified incorrectly, no changes are made. The paragraph spacing is measured in internal units (**InternalUnit**).

See Also	GetParagraphIndex method, Paragraph method, ParagraphsNum method, RemoveParagraph method, SetParaAfterSpacing method, SetParaBeforeSpacing method, SetParaFirstInd method, SetParaHAlign method, SetParaLineSpacing method, SetParaRightInd method, Paragraph object

SetParaLineSpacing Method

SetParaLineSpacing Method

Sets the line spacing for the specified paragraph of the shape.

Applies to: Shape object

Syntax

object.**SetParaLineSpacing** (*iFrom*, *iTo*, *LineSpacing*)

The **SetParaLineSpacing** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character of the paragraph.
iTo	Required. An expression that returns a **Long** value. The index of the last character of the paragraph.
LineSpacing	Required. An expression that returns a **Single** value. The line spacing value for the specified paragraph.

Remarks

Note, that if the paragraph collection of the shape doesn't contain the paragraph that corresponds to the character sequence within the *iFrom* to *iTo* range, the **SetParaLineSpacing** method adds a new paragraph and sets the specified line spacing to it.

If the *iFrom* and *iTo* parameters were specified incorrectly, no changes are made. The paragraph spacing is measured in internal units (**InternalUnit**).

See Also	GetParagraphIndex method, Paragraph method, ParagraphsNum method, RemoveParagraph method, SetParaAfterSpacing method, SetParaBeforeSpacing method, SetParaFirstInd method, SetParaHAlign method, SetParaLeftInd method, SetParaRightInd method, Paragraph object

SetParaRightInd Method

SetParaRightInd Method

Sets the distance the paragraph's text is indented from the right edge of the text block.

Applies to: Shape object

Syntax

object.**SetParaRightInd** (*iFrom*, *iTo*, *RightInd*)

The **SetParaRightInd** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
iFrom	Required. An expression that returns a **Long** value. The index of the first character of the paragraph.
iTo	Required. An expression that returns a **Long** value. The index of the last character of the paragraph.
RightInd	Required. An expression that returns a **Single** value. The paragraph indent from the right edge of the text box.

Remarks

Note, that if the paragraph collection of the shape doesn't contain the paragraph that corresponds to the character sequence within the *iFrom* to *iTo* range, the **SetParaLeftInd** method adds a new paragraph and applies the specified right indent to it.

If the *iFrom* and *iTo* parameters were specified incorrectly, no changes are made. The paragraph spacing is measured in internal units (**InternalUnit**).

See Also	GetParagraphIndex method, Paragraph method, ParagraphsNum method, RemoveParagraph method, SetParaAfterSpacing method, SetParaBeforeSpacing method, SetParaFirstInd method, SetParaHAlign method, SetParaLeftInd method, SetParaLineSpacing method, Paragraph object

SetPenColor Method

SetPenColor Method

Establishes color of lines of object (shape) for the current style of the document.

Applies to: Style object

Syntax

object.**SetPenColor** (*r*, *g*, *b*)

The **SetPenColor** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Style** object.
r	Required. An expression that returns a **Byte** value. Red color component in the RGB format. A range of admissible values from 0 to 255.
g	Required. An expression that returns a **Byte** value. Green color component in the RGB format. A range of admissible values from 0 to 255.
b	Required. An expression that returns a **Byte** value. Blue color component in the RGB format. A range of admissible values from 0 to 255.

Remarks

For obtaining color of lines it is possible to use **FillPenColor** property of object of **Style** which also is the tabular **CDPT_STYLED_LINECOLOR** parameter of object.

See Also	Style object, FillColor property, FillPatColor property, PenColor property, ShadowColor property, ShadowPatColor property, SetFillColor method, SetFillPatColor method, SetShadowColor method, SetShadowPatColor method.

SetPropertyFormula Method

SetPropertyFormula Method

Sets a table formula for a property. Returns **True** if the formula has been assigned correctly (doesn't contain errors), otherwise returns **False**.

Applies to objects: Shape, ServObj

Syntax

[[**Let**]*ret* =]*object*.**SetPropertyFormula**(*formulaStr*, *propTag* [, *num*[, *geom*]])

The **SetPropertyFormula** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
formulaStr	Required. An expression that returns a **String** value. The formula to be assigned to the property.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. It is used only for the objectShape. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. It is used only for the objectShape. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.
ret	Optional. A variable that gets the value returned by the method.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object and **ServObj** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

SetRectEmpty Method

SetRectEmpty Method

Resets to zero the properties of a DRect object.

Applies to objects: DRect

Syntax

object.**SetRectEmpty** ()

The **SetRectEmpty** method syntax has these Elements:

Element	Description
object	A reference to an instance of the object.

Example

This example uses the **SetRectEmpty** method.

```
' Create an instance of the object
Dim MyObject as new DRect
' set left,top,right,bottom properties
MyObject.SetRect(100,100,200,300)
' Reset the left,top,right,bottom properties to zero
MyObject.SetRectEmpty()
```

See Also DRect Object

SetRect Method

SetRect Method

Sets the left, top, right, bottom coordinates of an instance of the object.

Applies to objects: DRect

Syntax

object.**SetRect**(*left*, *top*, *right*, *bottom*)

The **SetRect** statement syntax has these Elements:

Element	Description
object	A reference to an instance of the object.
left, top, right, bottom	The coordinates of the rectanlge, Double values.

Remarks

This method offers a faster way of setting the coordinates of a rectangle, rather then setting them for each property separately.

Example

```
' Create an instance of the object
Dim MyObject as new DRect
' Set left,top,right,bottom properties
MyObject.SetRect(100,100,1000,1000)
```

See Also DRect Object

SetRGB Method

Sets the color scheme to RGB and initializes the color components with the specified values.

Applies to: Color object, ColorEntry object

Syntax

object.**SetRGB** (*red*, *green*, *blue*)

The **SetRGB** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object in the **Applies to** list.
red	Required. An expression that returns an **Integer** value. The value of the red component.
green	Required. An expression that returns an **Integer** value. The value of the green component.
blue	Required. An expression that returns an **Integer** value. The value of the blue component.

Remarks

After the **SetRGB** method has been called, the color is converted to the RGB format regardless of its previous format. The values of the *red*, *green* and *blue* can be in the range of **0** to **255**, and are used to set respective components of the color.

Example

This example contains a document-level script. It demonstrates how to change the fill color of a rectanlge in RGB format.

```
dim s as shape
' Create a Shape object
s = thisDoc.ActivePage.DrawRect(100,100,1000,1000)
s.FillColor.SetRGB(30,230,178)   ' Change the Shape's fill color in RGB format
s.PropertyChanged(CDPT_FILLCOLOR)
```

See Also SetCMYK method

SetShadowColor Method

SetShadowColor Method

Establishes color of a shadow of object (shape) for the current style of the document.

Applies to: Style object

Syntax

object.**SetShadowColor** (*r*, *g*, *b*)

The **SetShadowColor** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Style** object.
r	Required. An expression that returns a **Byte** value. Red color component in the RGB format. A range of admissible values from 0 to 255.
g	Required. An expression that returns a **Byte** value. Green color component in the RGB format. A range of admissible values from 0 to 255.
b	Required. An expression that returns a **Byte** value. Blue color component in the RGB format. A range of admissible values from 0 to 255.

Remarks

For obtaining color of a shadow it is possible to use **ShadowColor** property of object of **Style** which also is the tabular **CDPT_STYLED_SHADOWBACKGND** parameter of object.

See Also	Style object, FillColor property, FillPatColor property, PenColor property, ShadowColor property, ShadowPatColor property, SetFillColor method, SetFillPatColor method, SetPenColor method, SetShadowPatColor method.

SetShadowPatColor Method

SetShadowPatColor Method

Establishes color of a pattern (template) of a shadow of object (shape) for the current style of the document.

Applies to: Style object

Syntax

object.**SetShadowPatColor** (*r*, *g*, *b*)

The **SetShadowPatColor** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Style** object.
r	Required. An expression that returns a **Byte** value. Red color component in the RGB format. A range of admissible values from 0 to 255.
g	Required. An expression that returns a **Byte** value. Green color component in the RGB format. A range of admissible values from 0 to 255.
b	Required. An expression that returns a **Byte** value. Blue color component in the RGB format. A range of admissible values from 0 to 255.

Remarks

For obtaining color of a pattern (template) of a shadow it is possible to use **ShadowPatColor** property of object of **Style** which also is the tabular **CDPT_STYLED_SHADOWFOREGND** parameter of object.

See Also	Style object, FillColor property, FillPatColor property, PenColor property, ShadowColor property, ShadowPatColor property, SetFillColor method, SetFillPatColor method, SetPenColor method, SetShadowColor method.

SetShape Method

SetShape Method

Copies a shape into the specified master object. Alters the contents of the master object (the **Shape** property).

Applies to: Master object

Syntax

object.**SetShape** (*shapeSrc*)

The **SetShape** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Master** object.
countRet	Optional. A **Long** type variable.

Remarks

If the shape specified by *shapeSrc* couldn't be copied, the **SetShape** method doesn't change the shape in the **Shape** property. If the shape has been copied successfully, the shape contained in this master object becomes identical to the one specified by *shapeSrc*.

See Also Shape property, Equal method, SetIcon method, Shape object

SetSingleProperty Method

SetSingleProperty Method

Sets the value of a **Single** type property.

Applies to objects: Shape

Syntax

object.**SetSingleProperty**(*data*, *propTag* [, *num*[, *geom*]])

The **SetSingleProperty** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the Shape object.
data	Required. An expression that returns a **Single** value. The value to be set to the property.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.

num	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

SetStringProperty Method

SetStringProperty Method

Sets the value of a **String** type property.

Applies to objects: Shape, ServObj

Syntax

object.**SetStringProperty**(*data*, *propTag* [, *num*[, *geom*]])

The **SetStringProperty** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of an object from the **Applies to** list.
data	Required. An expression that returns a **String** value. The value to be set to the property.
propTag	Required. An expression that returns a **Long** value. A tag that identifies the property of the object.
num	Optional. It is used only for the objectShape. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from collections of the object.
geom	Optional. It is used only for the objectShape. An expression that returns a **Long** value. An additional identifying argument. It's used for specifying properties from geometry collections of the object.

Remarks

ConceptDraw shapes are described by sets of properties which can have so called table formulas. Properties can be viewed or edited in the shape parameter table, called from a menu or using the **F3** key in ConceptDraw. Each property is described by its value and a table formula.

This method is one of the methods of the **Shape** object and **ServObj** object, which allow to access the properties from a ConceptDraw Basic script. Such methods use three arguments for choosing the needed property: *propTag*, *num*, *geom*. Here, *propTag* is the tag that corresponds to the name of the property, and *num* and *geom* indicate the numbers of the properties in the collections. ConceptDraw Basic has a set of constants that define all possible property tags.

See Also	GetByteProperty method, GetBooleanProperty method, GetIntegerProperty method, GetLongProperty method, GetSingleProperty method, GetDoubleProperty method, GetStringProperty method, ColorProperty method, SetByteProperty method, SetBooleanProperty method, SetIntegerProperty method, SetLongProperty method, SetSingleProperty method, SetDoubleProperty method, SetStringProperty method, IsDefaultFormula method, IsNullFormula method, GetPropertyFormula method, SetPropertyFormula method, SetDefaultFormula method, SetNullFormula method, RecalcProperty method, PropertyChanged method

SetStyle Method

SetStyle Method

Sets a style to the shape. The style is specified by its name (the **Name** property).

Applies to: Shape object

Syntax

[[**Let**] *booleanRet* =] *object*.**SetStyle** (*styleName*)

The **SetStyle** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
styleName	Required. An expression that returns a **String** value. The name of the style (the **Name** property).
shapeRet	Optional. A **Boolean** type variable.

Remarks

If *styleName* is an empty string, the **SetStyle** method applies a null style (No Style) to the shape and returns **True**. If there is not style with the *styleName* name in the style collection of the document, which owns *object*, the **SetStyle** method doesn't change the current style and returns false. Otherwise, the method applies the new style to the shape and returns **True**.

SetWindowRect Method

SetWindowRect Method

Sets the size and position of the window.

Applies to: Window object

Syntax

object.**SetWindowRect** (*left*, *top*, *width*, *height*)

The **SetWindowRect** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Window** object.

left	Required **Long**. The x-coordinate of the left upper corner of the window.
top	Required **Long**. The y-coordinate of the left upper corner of the window.
width	Required **Long**. The distance from the left side to the right side of the window.
height	Required **Long**. The distance from the top side to the bottom side of the window.

Remarks

Note, that the size and position of the window are measured in screen pixels, and the coordinate origin is located in the left top corner of the parent window frame. Use the **SetWindowRect** to change the size and position of the window. To get the current size and position of the window, use the **Left**, **Top**, **Height** and **Width** properties.

See Also	Left property, Top property, Height property, Width property

ShapeByID Method

ShapeByID Method

Searches for a shape with the specified ID (**ID** property) in the shape collection of the group/page. Returns an instance of the **Shape** object that corresponds to the found shape.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**ShapeByID** (*shapeID*)

The **ShapeByID** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
shapeID	Required. An expression that returns a **Long** value. The ID of the shape to be found.

shapeRet	Optional. A **Shape** type variable.

Remarks

If there is no shape with the *shapeID* ID in the collection, the **ShapeByID** method returns **Nothing**.

See Also	ID property, ShapeBySubID method, RemoveAllShapes method, RemoveShape method, RemoveShapeByID method, ReorderShape method, ReorderShapeByID method, Shape method, ShapesNum method

ShapeBySubID Method

ShapeBySubID Method

Searches for a shape with the specified SubID (**SubID** property) in the shape collection of the group/page. Returns an instance of the **Shape** object that corresponds to the found shape.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**ShapeBySubID** (*shapeSubID*)

The **ShapeBySubID** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
shapeSubID	Required. An expression that returns a **Long** value. The SubID of the shape to be found.
shapeRet	Optional. A **Shape** type variable.

Remarks

If there is no shape with the *shapeSubID* SubID in the collection, the **ShapeBySubID** method returns **Nothing**.

Example

This example contains a shape-level script. In the first example the object (shape) having SubID equal 4, is looked for on page. In the second example the object (shape) having SubID equal 4, is looked for in group of objects.

```
dim sh as Shape
sh = thisPage.ShapeBySubID(4)
or
sh = thisShape.ShapeBySubID(4)
```

See Also	SubID property, ShapeByID method, RemoveAllShapes method, RemoveShape method, RemoveShapeByID method, ReorderShape method, ReorderShapeByID method, Shape method, ShapesNum method

ShapesNum Method

ShapesNum Method

Returns the number of shapes in the shape collection of the group/page.

Applies to: Page object, Shape object

Syntax

[[**Let**] *countRet* =] *object*.**ShapesNum** ()

The **ShapesNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an object in the **Applies to** list.
countRet	Optional. A **Long** type variable.

Remarks

If there are no shapes in the shape collection of the group/page, the **ShapesNum** method returns **0**.

See Also	RemoveAllShapes method, RemoveShape method, RemoveShapeByID method, ReorderShape method, ReorderShapeByID method, Shape method, ShapeByID method

Shape Method

Shape Method

Returns an instance of the **Shape** object, that corresponds to a shape with the specified index in the shape collection of the group/page.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**Shape** (*index*)

The **Shape** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns an instance of an object from the **Applies to** list.
index	Required. An expression that returns a **Long** value. The index of the shape in the shape collection of the group/page.
shapeRet	Optional. A **Shape** type variable.

Remarks

If *index* is less than **1** or greater than the number of shapes in the group/page, the **Shape** method returns **Nothing**. Use the **ShapesNum** method to find out the number of shapes in the group/page.

See Also RemoveAllShapes method, RemoveShape method, RemoveShapeByID method, ReorderShape method, ReorderShapeByID method, ShapeByID method, ShapesNum method

SplineStart Method

SplineStart Method

Starts creating a new spline segment. Returns a **Shape** object that corresponds to the shape in which the spline segment has been built.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**SplineStart** (*xBegin*, *yBegin*, *xA*, *yB*)

The **SplineStart** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
xStart	Required. An expression that returns a **Double** value. The X-coordinate of the begin point of the spline.
yStart	Required. An expression that returns a **Double** value. The Y-coordinate of the begin point of the spline.
xA	Required. An expression that returns a **Double** value. The X-coordinate of the guiding point.
yB	Required. An expression that returns a **Double** value. The Y-coordinate of the guiding point.
shapeRet	Optional. A **Shape** type variable.

Remarks

The SplineStart method adds a new spline start segment to the shape. The spline start segment is described by two points: the begin point of the spline (*xStart*, *yStart*) and the guiding point (*xA*, *yB*).

If *object* is a page or a group, the **SplineStart** method adds the spline start segment to the current Basic shape and returns a **Shape** object that corresponds to that shape. If the method was called prior to the **BeginShape** method or after the **EndShape** method, the **SplineStart** method doesn't create anything and returns **Nothing**.

If *object* is a simple shape, the **SplineStart** method adds to *object* a new geometry that contains the spline start segment with the specified coordinates and returns *object*.

The coordinates of the points are in the coordinate system of the shape, group or the page to which *object* corresponds. The coordinates are measured in internal units (**InternalUnit**).

See Also ArcTo method, BeginShape method, EndShape method, LineTo method, MoveTo method, SplineTo method

SplineTo Method

SplineTo Method

Builds a spline segment in a shape. Returns an instance of the **Shape** object, corresponding to the shape where the spline segment has been built.

Applies to: Page object, Shape object

Syntax

[[**Set**] *shapeRet* =] *object*.**SplineTo** (*xKnot*, *yKnot*, *xA*, *yB*)

The **SplineTo** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an object in the **Applies to** list.
xKnot	Required. An expression that returns a **Double** value. The X-coordinate of the end point of the spline segment.
yKnot	Required. An expression that returns a **Double** value. The Y-coordinate of the end point of the spline segment.
xA	Required. An expression that returns a **Double** value. The X-coordinate of the end guiding point of the spline segment.
yB	Required. An expression that returns a **Double** value. The Y-coordinate of the end guiding point of the spline segment.
shapeRet	Optional. A **Shape** type variable.

Remarks

The **SplineTo** method adds to the shape a new spline segment, described by four points: the begin point (the X and Y fields of the previous segment), the begin guiding point (the A and B fields of the previous segment), the end point (specified by the *xKnot* and *yKnot* parameters), the end guiding point (*xA* and *xB* parameters). Note, that a spline segment can only be added to a start spline segment or previous spline segment. That is, the **SplineStart** method must be called prior to the **SplineTo** method.

If *object* is a page or a group, the **SplineTo** method adds the spline start segment to the current Basic shape and returns a **Shape** object that corresponds to that shape. If the method was called prior to the **BeginShape** method or after the **EndShape** method, the **SplineTo** method doesn't create anything and returns **Nothing**.

If *object* is a simple shape, the **SplineTo** method adds a new spline segment to this shape and returns *object*.

The coordinates of the points are in the coordinate system of the shape, group or the page to which *object* corresponds. The coordinates are measured in internal units (**InternalUnit**).

See Also ArcTo method, BeginShape method, EndShape method, LineTo method, MoveTo method, SplineStart method

StartRebuild Method

StartRebuild Method

Informs the ConceptDraw Engine about the beginning of changing some properties of the shapes of the document.

Applies to: Document object

Syntax

object.**StartRebuild** ()

The **StartRebuild** method syntax has these Elements:

Element	Description
object	Required. An expression that returns an instance of the **Document** object.

Remarks

In order to inform the ConceptDraw application about the end of changing groups of properties of the shapes, and re-calculate all modified properties, use the **EndRebuild** method. Such scheme of changing shape properties applies when you need to change several properties of shapes without re-calculating all dependent properties after each change. In this case all properties are re-calculated just once on calling the **EndRebuild** method.

See Also EndRebuild method, UpdateAllViews method

StepBack Method

StepBack Method

Moves the object (shape) back by one position in the collection of objects (shapes) of the parent group. Returns the index of the object (shape) in a collection of objects (shapes) of the parent group.

Applies to: Shape object

Syntax

[[Let] *index* =] *object*. **StepBack** ()

The **StepBack** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
index	Optional. A **Long** type variable.

Remarks

If the *object* is an object directly on the document page, then the parent of this object is a page (Property **Page).** If an object is placed in a group, then its parent is a group of objects. The numbering of objects starting with 0. In case of error the method returns -1.

Example

```
dim index as Integer
index = thisShape.GetIndex()
trace index
index = thisShape.StepBack()
trace index
```

See Also	GetIndex method, Page property, Parent property, SendBack method, SendFront method, StepFront method

StepFront Method

StepFront Method

Moves the object (shape) by one position in the collection of objects (shapes) of the parent group. Returns the index of the object (shape) in a collection of objects (shapes) of the parent group.

Applies to: Shape object

Syntax

[[Let] *index* =] *object.* **StepFront** ()

The **StepFront** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
index	Optional. A **Long** type variable.

Remarks

If the *object* is an object directly on the document page, then the parent of this object is a page (Property **Page).** If an object is placed in a group, then its parent is a group of objects. The numbering of objects starting with 0. In case of error the method returns -1.

Example

```
dim index as Integer
index = thisShape.GetIndex()
trace index
index = thisShape.StepFront()
trace index
```

See Also	GetIndex method, Page property, Parent property, SendBack method, SendFront method, StepBack method

StyleByName Method

StyleByName Method

Searches for a style with the specified name (the **Name** property) in the style collection of the document. Returns a **Style** object, that corresponds to the found style.

Applies to: Document object

Syntax

[[**Set**] *styleRet* =] *object*.**StyleByName** (*styleName*)

The **StyleByName** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns a **Document** object.
styleName	Required. An expression that returns a **String** value. The name (the **Name** property) of the style to be found.
styleRet	Optional. A **Style** type variable.

Remarks

If there is no style with the specified *styleName* in the style collection of the document, the **StyleByName** method returns **Nothing**.

See Also AddStyle method, FindStyle method, RemoveStyle method, RemoveStyleByName method, RenameStyle method, Style method, StylesNum method, Style object

StylesNum Method

StylesNum Method

Returns the number of the styles in the style collection of the document.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**StylesNum** ()

The **StyelsNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression, that returns an instance of the **Document** object.
countRet	Optional. A **Long** type variable.

Remarks

If there are no styles in the style collection of the document, the **StylesNum** method returns **0**.

See Also AddStyle method, FindStyle method, RemoveStyle method, RemoveStyleByName method, RenameStyle method, Style method, StyleByName method, Style object

Style Method

Style Method

Returns an instance of the **Style** object by the specified ID of the style in the style collection of the document.

Applies to: Document object

Syntax

[[**Set**] *styleRet* =] *object*.**Style** (*index*)

The **Style** method syntax has these Elements:

Element	Description
object	Required. An expression, that returns a **Document** object.
index	Required. An expression that returns a **Long** value. Indicates the style index in the style collection of the document.
styleRet	Optional. A **Style** type variable.

Remarks

If *index* is less than **1** or greater than the number of styles in the style collection of the document, the **Style** method returns **Null**. Use the **StylesNum** method find out the number of styles in the document.

See Also AddStyle method, FindStyle method, RemoveStyle method, RemoveStyleByName method, RenameStyle method, StyleByName method, StylesNum method, Style object

TabStopsNum Method

TabStopsNum Method

Returns the number of tab stops in the specified text block.

Applies to: TextBlock object

Syntax

[[**Let**] *countRet* =] *object*.**TabStopsNum** ()

The **TabStopsNum** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **TextBlock** object.
countRet	Optional. A **Long** type variable.

Remarks

If the text block doesn't contain any tab stops, the method returns **0**.

Example

This example demonstrates using the **TabStopsNum** method. It assumes that the active page already contains the Shape with ID1, which has text, and probably one or more tab stops.

```
Dim s as Shape
s = thisDoc.ActivePage.ShapeByID(1)
' Display the number of tab stops.
trace s.TextBlock.TabStopsNum()
```

See Also AddTabStop method, RemoveTabStop method, TabStop method

TabStop Method

TabStop Method

Returns a **TabStop** object, that corresponds to a tab stop with the specified index in the tab stop collection of the text block.

Applies to: TextBlock object

Syntax

[[**Set**] *tabStopRet* =] *object*.**TabStop** (*index*)

The **TabStop** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **TextBlock** object.
index	Required. An expression that returns a **Long** value. The index of the tab stop in the tab stop collection of *object*'s text block.
tabStopRet	Optional. A **TabStop** type variable.

Remarks

If *index* is less than **1** or greater than the number of tab stops in the text block, the **TabStop** method returns **Nothing**. Use the **TabStopsNum** method to find out the number of tab stops in the text block.

See Also	AddTabStop method, RemoveTabStop method, TabStopsNum method, TabStop object

UnionRect Method

UnionRect Method

Calculates the coordinates of a rectangle with the least possible square enough to encompass to specified rectangles. Returns a Boolean value: FALSE, if the specified rectangles are empty, otherwise - TRUE.

Applies to objects: DRect

Syntax

[[**Let**] *res* =] *object*.**UnionRect** (*inRect1*, *inRect2*)

The **UnionRect** method syntax has these Elements:

Element	**Description**
object	A reference to an instance of the object.

inRect1, *inRect2*	References to instances of the DRect object.
res	Variable of a **Boolean** type.

Remarks

If *inRect1* and *inRect2* have zero square, this method returns False, and the properties of the instance of the *object* object, for which the method was called, are reset to zero.

Example

```
Dim outRect as new DRect, inRect1 as new DRect, inRect2 as new DRect, res as
Boolean
inRect1.SetRect(100,100,200,200)
inRect2.SetRect(200,200,400,400)
'outRect properties take these values: 100,100,400,400
res = outRect.UnionRect(inRect1,inRect2)   ' return TRUE
```

See Also

DRect Object

UpdateAllViews Method

UpdateAllViews Method

Re-draws all document windows.

Applies to: Document object

Syntax

object.**UpdateAllViews** ()

The **UpdateAllViews** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.

See Also EndRebuild method, StartRebuild method

VariablesNum Method

VariablesNum Method

Returns the number of user-defined variables in the shape.

Applies to: Shape object

Syntax

[[**Let**] *coutnRet* =] *object*.**VariablesNum** ()

The **VariablesNum** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
countRet	Optional. A **Long** type variable.

Remarks

If the shape doesn't contain any user-defined variable, the **VariablesNum** method returns **0**.

See Also AddVariable method, Variable method, RemoveVariable method, Variable object

Variable Method

Variable Method

Returns a **Variable** object that corresponds to a user-defined variable with the specified index in the user-defined variable collection of the shape.

Applies to: Shape object

Syntax

[[**Set**] *variableRet* =] *object*.**Variable** (*index*)

The **Variable** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
index	Required. An expression that returns a **Long** value. The index of the variable in the variable collection of the shape.
variableRet	Optional. A **Variable** type variable.

Remarks

If *index* is less than **1** or greater than the number of user-defined variables in the variable collection of the shape, the **Variable** method returns **Nothing**. Use the **VariablesNum** method find out the number of variables in the shape.

See Also	AddVariable method, VariablesNum method, RemoveVariable method, Variable object

ViewByID Method

ViewByID Method

Searches for a window with the specified ID (the **ID** property) in the window collection of the document. Returns a **Window** object that corresponds to the found document window.

Applies to: Document object

Syntax

[[**Let**] *windowRet* =] *object*.**ViewByID** (*viewID*)

The **ViewByID** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Document** object.

viewID	Required. An expression that returns a **Long** value. The ID (**ID** property) of the window to be found.
windowRet	Optional. A **Window** type variable.

Remarks

If there is no window with the *viewID* ID in the collection, the **ViewByID** method returns **Nothing**.

See Also FirstView method, NextView method, ViewsNum method, UpdateAllViews method

ViewsNum Method

ViewsNum Method

Returns the number of open windows for the specified ConceptDraw document.

Applies to: Document object

Syntax

[[**Let**] *countRet* =] *object*.**ViewsNum** ()

The **ViewsNum** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Document** object.
countRet	Optional. A **Long** type variable.

Remarks

The **ViewsNum** returns a value equal to or greater than **1**, as an open ConceptDraw document always has at least one window.

See Also	FirstView method, NextView method, ViewByID method, UpdateAllViews method

WPtoLP Method

WPtoLP Method

Converts the coordinates of the specified point from the world coordinate system to the local coordinate system of this shape.

Applies to: Shape object

Syntax

object.**WPtoLP** (*srcPoint*)

The **WPtoLP** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
srcPoint	Required. A **DPoint** type variable. The coordinates of the point.

Remarks

This method modifies the input argument *srcPoint* and uses it to return the resulting coordinates. The coordinates are measured in internal units (**InternalUnit**).

See Also	GPtoLp, LAtoWA method, LPtoGP method, LPtoWP method

XPathText Method

XPathText Method

Returns the text written in the specified XML file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **XPathText** *(dsIndex, xPathExpr, defVal)*

The **XPathText** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
xPathExpr	Required. An expression that returns a **String** value. XPATH expression.
defVal	Required. An expression that returns a **String** value. The default value.
ret	Optional. A **String** type variable.

Remarks

The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the data that resides in the XPATH expression "/ Localization / XPATHText" a second source of data sources in the collection of data object (shape).

```
dim res as String
res = thisShape.XPathText (2, "/ Localization / XPATHText", "Error")
trace res
```

A fragment of xml file:

```
<Localization Version="1">
<XPATHValue> 776 </ XPATHValue>
<XPATHValueD> 776.68 </ XPATHValueD>
<XPATHText> Reed the XPATH Text </ XPATHText>
</ Localization>
```

DataSource object , FileText , XPathValue , XPathValueD

See Also

XPathValueD Method

Returns the value of the specified XML file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **XPathValueD** *(dsIndex, xPathExpr, defVal)*

The **XPathValueD** method syntax has these Elements:

Element	Description
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
xPathExpr	Required. An expression that returns a **String** value. XPATH expression.
defVal	Required. An expression that returns a **Double** value. The default value.
ret	Optional. A **Double** type variable.

Remarks

The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the data that resides in the XPATH expression "/ Localization / XPATHValueD" a second source of data sources in the collection of data object (shape).

```
dim res as Double
res = thisShape.XPathValueD (2, "/ Localization / XPATHValueD", -1.5)
trace res
```

A fragment of xml file:

```
<Localization Version="1">
<XPATHValue> 776 </ XPATHValue>
<XPATHValueD> 776.68 </ XPATHValueD>
<XPATHText> Reed the XPATH Text </ XPATHText>
</ Localization>
```

DataSource object , XPathText , XPathValue

See Also

XPathValue Method

Returns the integer value from the specified XML file data source object (shape).

Applies to: Shape object

Syntax

[[Let] *ret* =] *object.* **XPathValue** *(dsIndex, xPathExpr, defVal)*

The **XPathValue** method syntax has these Elements:

Element	**Description**
object	Required. An expression that returns a **Shape** object.
dsIndex	Required. An expression that returns a **Long** value. Index data source in the collection of data source object (shape).
xPathExpr	Required. An expression that returns a **String** value. XPATH expression.
defVal	Required. An expression that returns a **Long** value. The default value.
ret	Optional. A **Long** type variable.

Remarks

The numbering of the data sources in the collection of data sources, the object starts at 1. The default value is set out in the case of addressing the range of the table or in the case of missing data or not corresponding to the data type and return type.

Example

Getting the data that resides in the XPATH expression "/ Localization / XPATHValue" a second source of data sources in the collection of data object (shape).

```
dim res as Long
res = thisShape.XpathValue (2, "/ Localization / XPATHValue", -1)
trace res
```

A fragment of xml file:

```
<Localization Version="1">
<XPATHValue> 776 </ XPATHValue>
<XPATHValueD> 776.68 </ XPATHValueD>
<XPATHText> Reed the XPATH Text </ XPATHText>
</ Localization>
```

DataSource object , XPathText , XPathValueD

See Also

ConceptDraw access Objects Constants

Import / Export Constants

These constants are used in the Import/Export methods (such as Import method, Export method).

Constant	Value	Import	Export	Description
cdf_UNKNOWN	0	-	-	Means unknown format of file.
cdf_CDD	1	Yes	Yes	ConceptDraw V document file format.
cdf_CDT	2	Yes	Yes	ConceptDraw V template file format.
cdf_CDL	3	Yes	Yes	ConceptDraw V library file format.
cdf_CDW	4	Yes	Yes	ConceptDraw V workspace file format.
cdf_CDD1X	5	Yes	Yes	ConceptDraw 1.x document file format.
cdf_CDT1X	6	Yes	Yes	ConceptDraw 1.x template file format.
cdf_CDL1X	7	Yes	Yes	ConceptDraw 1.x library file format.
cdf_CDW1X	8	Yes	Yes	ConceptDraw 1.x workspace file format.
cdf_CDB	9	No	No	ConceptDraw Basic script source file format.
cdf_BMP	10	Yes	Yes	Bitmap file format.
cdf_DIB	11	Yes	Yes	Device-independent bitmap file format.
cdf_DCM	12			
cdf_GIF	13	Yes	Yes	Graphics Interchange format.
cdf_ICO	14	Yes	Yes	Windows icon file format.
cdf_ICON	15	Yes	Yes	Windows icon file format.
cdf_JPEG	16	Yes	Yes	Joint Photographic Experts Group file format.
cdf_JPG	17	Yes	Yes	Joint Photographic Experts Group file format.
cdf_PNG	18	Yes	Yes	Portable Network Graphics file format.
cdf_PCD	19	Yes	Yes	
cdf_PCDS	20			
cdf_PCX	21	Yes	Yes	
cdf_SGI	22	Yes	Yes	
cdf_RAS	23	Yes	Yes	
cdf_SUN	24			
cdf_TGA	25	Yes	Yes	
cdf_ICB	26			
cdf_VDA	27			
cdf_VST	28			

cdf_TIF	29	Yes	Yes	Tag Image file format.
cdf_TIFF	30	Yes	Yes	Tag Image file format.
cdf_WPG	31	Yes	Yes	
cdf_XBM	32	Yes	Yes	
cdf_XPM	33	Yes	Yes	
cdf_PCT	34	Yes	Yes	
cdf_DXF	35	Yes	Yes	
cdf_HTM	36	No	Yes	Hypertext Markup Language file format.
cdf_HTML	37	No	Yes	Hypertext Markup Language file format.
cdf_EPS	38	No	Yes	Encapsulated postscript file format
cdf_CDX	39	Yes	Yes	XML for ConceptDraw file format.
cdf_OUTLINE	40	Yes	Yes	ConceptDraw outline file format. It is text format file.
cdf_FLOWDATA	41	Yes	Yes	ConceptDraw flowdata file format.
cdf_PPT	42	Yes	Yes	MS PowerPoint file format
cdf_EMF	43	Yes	Yes	Enhanced Metafile format.
cdf_WMF	44	Yes	Yes	Windows Metafile format.
cdf_PAL	45			
cdf_SWF	46	No	Yes	Macromedia Flash format.
cdf_PDF	47	No	Yes	
cdf_PSD	48	Yes	Yes	Adobe Photoshop Drawing format.
cdf_VDX	49	Yes	No	Microsoft Visio XML format.
cdf_SVG	50	No	Yes	Scalable Vector Graphic.
cdf_PICT	51	Yes	No	Macintosh PICT.
cdf_CDOCMD	52	Yes	YES	Conceptdraw Office command file format.
cdf_CDLX	53	Yes	Yes	ConceptDraw XML Libraries file format.
cdf_CDTX	54	Yes	Yes	ConceptDraw XML Template file format.

Property Tag Constants

Constant	**Value**
CDPT_WIDTH	1
CDPT_HEIGHT	2
CDPT_ANGLE	3
CDPT_GPINX	4
CDPT_GPINY	5
CDPT_FLIPX	6
CDPT_FLIPY	7

CDPT_LPINX	8
CDPT_LPINY	9
CDPT_BEGINX	10
CDPT_BEGINY	11
CDPT_ENDX	12
CDPT_ENDY	13
CDPT_GEOMETRY_X	14
CDPT_GEOMETRY_Y	15
CDPT_GEOMETRY_A	16
CDPT_GEOMETRY_B	17
CDPT_GEOMETRY_C	18
CDPT_GEOMETRY_D	19
CDPT_GEOMETRY_VISIBLE	20
CDPT_GEOMETRY_FILLED	21
CDPT_TEXTWIDTH	26
CDPT_TEXTHEIGHT	27
CDPT_TEXTANGLE	28
CDPT_TEXTPINX	29
CDPT_TEXTPINY	30
CDPT_TEXTGPINX	31
CDPT_TEXTGPINY	32
CDPT_VALIGN	33
CDPT_TOPMARGIN	34
CDPT_BOTTOMMARGIN	35
CDPT_LEFTMARGIN	36
CDPT_RIGHTMARGIN	37
CDPT_TEXTBKGND	38
CDPT_DEFTABSTOP	39
CDPT_TABALIGN	40
CDPT_TABPOS	41
CDPT_TEXT	42
CDPT_LINEPATTERN	43
CDPT_LINEWEIGHT	44
CDPT_LINECOLOR	45
CDPT_LINEBEGIN	46
CDPT_LINEEND	47
CDPT_LINEENDSIZE	48
CDPT_FILLPATTERN	49

CDPT_FILLPATCOLOR	50
CDPT_FILLCOLOR	51
CDPT_SHADOWPATTERN	52
CDPT_SHADOWPATCOLOR	53
CDPT_SHADOWCOLOR	54
CDPT_LOCKWIDTH	55
CDPT_LOCKHEIGHT	56
CDPT_LOCKMOVEX	57
CDPT_LOCKMOVEY	58
CDPT_LOCKASPECT	59
CDPT_LOCKCALCWH	60
CDPT_LOCKROTATE	61
CDPT_LOCKDELETE	62
CDPT_LOCKBEGIN	63
CDPT_LOCKEND	64
CDPT_LOCKVERTEX	65
CDPT_LOCKFLIPX	66
CDPT_LOCKFLIPY	67
CDPT_SHOWSHAPEHANDLES	68
CDPT_SHOWCONTROLHANDLES	69
CDPT_SHOWALIGNBOX	70
CDPT_NONPRINTING	71
CDPT_RESIZEBEHAVIOUR	72
CDPT_SHOWTEXT	73
CDPT_VARIABLE_X	74
CDPT_VARIABLE_Y	75
CDPT_CONTROL_X	76
CDPT_CONTROL_Y	77
CDPT_CONTROL_XDYN	78
CDPT_CONTROL_YDYN	79
CDPT_CONTROL_XBEHAVIOUR	80
CDPT_CONTROL_YBEHAVIOUR	81
CDPT_CONTROL_COMMENT	82
CDPT_CONNECT_X	83
CDPT_CONNECT_Y	84
CDPT_CHAR_FONT	85
CDPT_CHAR_SIZE	86
CDPT_CHAR_COLOR	87

CDPT_CHAR_STYLE	88
CDPT_CHAR_POS	90
CDPT_CHAR_LANGUAGE	91
CDPT_CHAR_SPACING	92
CDPT_CHAR_HYPERLINK	93
CDPT_PARA_FIRSTIND	94
CDPT_PARA_LEFTIND	95
CDPT_PARA_RIGHTIND	96
CDPT_PARA_HALIGN	97
CDPT_PARA_BEFORESPACING	99
CDPT_PARA_AFTERSPACING	100
CDPT_PARA_LINESPACING	101
CDPT_ACTION_ACTION	102
CDPT_ACTION_MENU	103
CDPT_ACTION_PROMPT	104
CDPT_ACTION_CHECKED	105
CDPT_ACTION_DISABLED	106
CDPT_CUSTOM_LABEL	107
CDPT_CUSTOM_PROMPT	108
CDPT_CUSTOM_TYPE	109
CDPT_CUSTOM_FORMAT	110
CDPT_CUSTOM_VALUE	111
CDPT_CUSTOM_INVISIBLE	112
CDPT_CUSTOM_VERIFY	113
CDPT_CONNECTOBJBEGIN	129
CDPT_CONNECTOBJEND	130
CDPT_CONNECTTYPEBEGIN	131
CDPT_CONNECTTYPEEND	132
CDPT_TEXTFLIPX	133
CDPT_TEXTFLIPY	134
CDPT_LAYER	142
CDPT_HYPERLINK	143
CDPT_DBLCLICK	144
CDPT_DBLCLICKACTION	145
CDPT_ROUNDCORNERS	148
CDPT_CONNECTMODE	149
CDPT_CONNECTORKNEELIMIT	150
CDPT_CONNECTBYPASSGROUPS	151

CDPT_EVENTPAGESREORDER	152
CDPT_EVENTPAGESCOUNT	153
CDPT_EVENTIDLE	154
CDPT_EVENTTIMER	155
CDPT_EVENTLOAD	156
CDPT_CHARPROPEVENT	157
CDPT_EVENTFILENAME	158
CDPT_LINEALPHA	159
CDPT_FILLFOREGNDALPHA	160
CDPT_FILLBACKGNDALPHA	161
CDPT_SHADOWFOREGNDALPHA	162
CDPT_SHADOWBACKGNDALPHA	163
CDPT_TEXTBKGNDALPHA	164
CDPT_CHAR_ALPHA	165
CDPT_FILLTEXTURE	166
CDPT_REGULAR_PROPS_NUM	167
CDPT_STYLED_LINEPATTERN	210
CDPT_STYLED_LINEWEIGHT	211
CDPT_STYLED_LINECOLOR	212
CDPT_STYLED_BEGINARROW	213
CDPT_STYLED_ENDARROW	214
CDPT_STYLED_ARROWSIZE	215
CDPT_STYLED_FILLPATTERN	216
CDPT_STYLED_FILLFOREGND	217
CDPT_STYLED_FILLBACKGND	218
CDPT_STYLED_SHADOWPATTERN	219
CDPT_STYLED_SHADOWFOREGND	220
CDPT_STYLED_SHADOWBACKGND	221
CDPT_STYLED_CHAR_FONT	222
CDPT_STYLED_CHAR_SIZE	223
CDPT_STYLED_CHAR_COLOR	224
CDPT_STYLED_CHAR_STYLE	225
CDPT_STYLED_CHAR_POS	226
CDPT_STYLED_CHAR_SET	227
CDPT_STYLED_CHAR_SPACING	228
CDPT_STYLED_PARA_FIRSTIND	229
CDPT_STYLED_PARA_LEFTIND	230
CDPT_STYLED_PARA_RIGHTIND	231

CDPT_STYLED_PARA_HALIGN	232
CDPT_STYLED_PARA_BEFOREIND	233
CDPT_STYLED_PARA_AFTERIND	234
CDPT_STYLED_PARA_BETWEENLN	235
CDPT_STYLED_VERTICALALIGN	236
CDPT_STYLED_TOPMARGIN	237
CDPT_STYLED_BOTTOMMARGIN	238
CDPT_STYLED_LEFTMARGIN	239
CDPT_STYLED_RIGHTMARGIN	240
CDPT_STYLED_TEXTBKGND	241
CDPT_STYLED_TXTDEFTABSTOP	242
CDPT_STYLED_LINEALPHA	243
CDPT_STYLED_FILLFOREGNDALPHA	244
CDPT_STYLED_FILLBACKGNDALPHA	245
CDPT_STYLED_SHADOWFOREGNDALPHA	246
CDPT_STYLED_SHADOWBACKGNDALPHA	247
CDPT_STYLED_CHAR_ALPHA	248
CDPT_STYLED_TEXTBKGNDALPHA	249
CDPT_DS_DATASOURCE	250
CDPT_DS_REFRESH_TIME	251
CDPT_DS_ACTION	252
CDPT_DS_VALID	253
CDPT_DS_ACTIVE	254
CDPT_DS_DATASOURCE_PATH	255
CDPT_DS_RELIABILITY	256
CDPT_DS_SHOW_WARNINGS	257
CDPT_DS_SHOW_ERRORS	258
CDPT_DSV_NAME	259
CDPT_DSV_VALUE	260
CDPT_DSV_TYPE	261
CDPT_DSV_VISIBLE	262
CDPT_DSV_OBJECT_TYPE	263
CDPT_DSV_SHOW_DIALOG	264
CDPT_LOCKTEXTBOUND	265
CDPT_LOCKGROUP	266
CDPT_LOCKFILL	267
CDPT_LOCKLINE	268
CDPT_RAPIDDRAW	269

CDPT_HIDEINSLIDESHOW	270
CDPT_RD_LIB_NAME	271
CDPT_RD_OBJ_NAME	272
CDPT_RD_ICON_NAME	273
CDPT_RD_LEFT_PLACING	274
CDPT_RD_RIGHT_PLACING	275
CDPT_RD_TOP_PLACING	276
CDPT_RD_BOTTOM_PLACING	277
CDPT_RD_CONNECTOR_TYPE	278
CDPT_RD_CONN_LIB_NAME	279
CDPT_RD_CONN_OBJ_NAME	280
CDPT_RD_AUTO_BALANCE	281
CDPT_RD_SPACING_X	282
CDPT_RD_SPACING_Y	283
CDPT_RD_START_CONN_POINT	284
CDPT_RD_END_CONN_POINT	285
CDPT_RD_SPACING_X_VERT_MOVE	286
CDPT_RD_SPACING_Y_HOR_MOVE	287
CDPT_RAPIDDRAW_OBJECT_BOUND	288
CDPT_RAPIDDRAW_TOP_AUTO_STEP	289
CDPT_RAPIDDRAW_LEFT_AUTO_STEP	290
CDPT_RAPIDDRAW_RIGHT_AUTO_STEP	291
CDPT_RAPIDDRAW_BOTTOM_AUTO_STEP	292
CDPT_RD_OBJECT_DESCRIPTION	293

Databases access Objects

About optionality of a collection object name

About optionality of a collection object name

At creation of a new collection object the parameter that defines the object being created is optional. For example,

```
Dim engine As dbEngine, wspace As Workspace
Set engine = new dbEngine
Set wspace = engine.CreateWorkspace()
```

Last code line creates a new instance of the object *Workspace* and adds it to the *Workspace* collection of *engine*. At that the object's name is not specified.

Getting access to such an object is possible by reference or by index in the collection. However, an unnamed object cannot be added to a database. To be added to a database it needs naming by using the property *Name*.

See Also CreateField Method, CreateIndex Method, CreateParameter Method, CreateProperty Method, CreateQueryDef Method, CreateRelation Method, CreateTableDef Method, CreateWorkspace Method

Connections Object

Connections Object

The **Connections** object is a collection of Connection objects and represents methods for controlling and accessing stored objects.

Methods

Count	Returns the number of objects, stored in the collection.
GetByName	Gets an object from the collection by its name.
GetByNumber	Gets an object from the collection by its number.
DeleteByName	Removes an object from the collection by its name.
DeleteByNum	Removes an object from the collection by its number.

Remarks

The **Connections** object belongs to the Workspace object and can be retrieved by calling the Connections method.

See Also Connection Object, Workspace Object, Connections Method

Connection Object

Connection Object

Describes a database connection. Provides access to transactions. Controls creating and using stored procedures and direct queries in the SQL language. An instance of this object can be retrieved by using the OpenConnection method of the Workspace collection or from the Connections collection.

Properties

Name	The name of the object for indentification in the collection.
DriverVersion	The ODBC driver version.
ConformanceLevel	The functionality level of the driver.
Transactions	Transaction support level.
IsolationLevels	A bit mask, describing available transaction isolation levels.
CursorTypes	A bit mask that describes available cursors.

Methods

CreateQueryDef	Creates a stored procedure in the SQL language. Returns a QueryDef object.
OpenRecordset	Is used for executing direct queries and stored procedures in the SQL language. Returns the result of the query - a Recordset object.
CreateProperty	Creates a Property object, that describes a user-defined property.
QueryDefs	Provides access to the QueryDefs collection.
Recordsets	Provides access to the Recordsets collection.
Properties	Provides access to the Properties collection.
BeginTrans	Begins a transaction. All subsequent actions on the database will form Element of this transaction.
CommitTrans	Applies all database changes since BeginTrans was called.
RollbackTrans	Ignores all changes in the database, occurred since BeginTrans was called. The database will be in the same state as before calling BeginTrans.
Close	Breaks a database connection.

Remarks

The DriverVersion, ConformanceLevel, Transactions, IsolationLevels and CursorTypes properties contain information, relating only to the given connection and have the **Read-Only** attribute.

The SQL code of a QueryDef object, created with the CreateQueryDef method, will be input in the database in the form of a stored procedure after calling the Append method of the QueryDefs collection. It can be executed by using the OpenRecordset method, with the name of the QueryDef object as the parameter.

Using transactions at this level assumes that transactions are applied to the given object exclusively.

If the transaction hasn't been closed before calling the Close method, the CommitTrans function will be called automatically for the changes to come into force.

See Also	Connections Object, Property Object, Properties Object, QueryDef Object, QueryDefs Object, Recordset Object, Recordsets Object, Append Method, OpenConnection Method

Databases Object

Databases Object

The **Databases** object represents a collection of Database objects and provides methods for accessing and controlling the stored objects.

Methods

Count	Returns the number of objects, stored in the collection.
GetByName	Gets an object from the collection by its name.
GetByNumber	Gets an object from the collection by its number.
DeleteByName	Removes an object from the collection by its name.
DeleteByNum	Removes an object from the collection by its number.

Remarks

The **Databases** object belongs to the Workspace object and can be retrieved by calling the Databases method.

See Also Database Object, Workspace Object, Databases Method

Database Object

Database Object

Describes a model of an open database. Controls transactions. Provides control over tables and links between tables. Allows to create and use stored procedures and direct SQL queries for controlling a database and getting information from it. An instance of this object can be retrieved by using the OpenDatabase method of the Workspace object or from the Databases collection.

Properties

Name	The name of the object for indentification in the collection.
DriverVersion	The ODBC driver version.
ConformanceLevel	The driver functionality level.
Transactions	Transaction support level.
IsolationLevels	A bit mask, describing available transaction isolation levels.
CursorTypes	A bit mask that describes available cursors.

Methods

CreateTableDef	Creates a new TableDef object, describing the data table.
CreateQueryDef	Creates a stored procedure in the SQL language. Returns a QueryDef object.
CreateRelation	Creates a Relation object, describing relationship between the tables.
OpenRecordset	Creates and executes a direct SQL query. Can also execute stored procedures. Returns the result of the query - a Recordset object.
CreateProperty	Creates a Property object, that describes a user-defined property.
TableDefs	Provides access to the TableDefs collection.
QueryDefs	Provides access to the QueryDefs collection.

Recordsets	Provides access to the Recordsets collection.
Relations	Provides access to the Relations collection.
Properties	Provides access to the Properties collection.
BeginTrans	Begins a transaction. All subsequent actions on the database will form Element of this transaction.
CommitTrans	Applies all database changes since BeginTrans was called.
RollbackTrans	Ignores all changes in the database, occurred since BeginTrans was called. The database will be in the same state as before calling BeginTrans.
Close	Breaks a database connection.

Remarks

The DriverVersion, ConformanceLevel, Transactions, CursorTypes and IsolationLevels properties contain information, relating only to the given connection and have the **Read-Only** attribute.

After calling the CreateTableDef method the new table will be created in the database only after the TableDef object, describing the table, has been completely formed, all table fields have been created, and the Append method of the TableDefs collection has been called.

Similarly, information about new relations between the tables will be added to the database only after calling the Append method of the Relations collection.

The SQL code of a QueryDef object, created with the CreateQueryDef method, will be input in the database in the form of a stored procedure after calling the Append method of the QueryDefs collection. It can be executed by using the OpenRecordset method, with the name of the QueryDef object as the parameter.

Using transactions at this level assumes that transactions are applied to the given object exclusively.

If the transaction hasn't been closed before calling the Close method, the CommitTrans function will be called automatically for the changes to come into force.

See Also	Databases Object, Property Object, Properties Object, QueryDef Object, QueryDefs Object, Recordset Object, Recordsets Object, Relation Object, Relations Object, TableDef Object, TableDefs Object, Workspace Object, Append Method, OpenDatabase Method

DBEngine Object

DBEngine Object

The **DBEngine** object is used for controlling the database access driver and Workspace objects.

Properties

DriverManager	The name of the used library.
DriverType	The driver type.

Methods

CreateProperty	Creates a Property object, that describes a user-defined property.
CreateWorkspace	Creates a Workspace object.
Properties	Provides access to the Properties collection.
Workspaces	Provides access to the Workspaces collection.

Remarks

By default, the DriverManager property equals **odbc32.dll** for Windows and **iODBC CFM Bridge** for Mac OS.

Presently, only ODBC drivers are fully supported. The DriverType property is reserved for future use. The changes you make in this property are ignored.

See Also Property Object, Properties Object, Workspace Object, Workspaces Object

Fields Object

Fields Object

The **Fields** object represents a collection of Field objects and provides methods for controlling and accessing the stored objects.

Methods

Count	Returns the number of objects, stored in the collection.
Append	Adds a new object to the collection.
GetByName	Gets an object from the collection by its name.
GetByNumber	Gets an object from the collection by its number.
DeleteByName	Removes an object from the collection by its name.
DeleteByNum	Removes an object from the collection by its number.
Refresh	Refreshes the object collection.

Remarks

The **Fields** object belongs to the Index, Recordset, Relation, TableDef objects and can be retrieved by calling the Fields method.

See Also Field Object, Index Object, Recordset Object, Relation Object, TableDef Object, Fields Method

Field Object

Field Object

Describes a data field. An instance of this object can be retrieved by using the CreateField method of the Index, TableDef and Relation objects or from the Fields collection.

Properties

Name	The name of the object for indentification in the collection.
Type	The field type.
Size	The field size.
Scale	The number of digits after the decimal point.
ForeignName	The name of the field, connected with the given relation.
Required	A flag, specifying whether this field is required.
SourceTable	The name of the table in the database, containing this field.
SourceField	The name of the field in the database.

AsString	Represents the contents of the field as a string.
AsLong	Represents the contents of the field as an integer number.
AsDouble	Represents the contents of the field as an real number.
AsBoolean	Represents the contents of the field as a boolean value.

Methods

GetMoreData	Checks if there are more data and gets the next portion.
CreateProperty	Creates a Property object, that describes a user-defined property.
Properties	Provides access to the Properties collection.

Remarks

The ForeignName property is used for creating relations between tables. It refers to the field, on which the external key of the related table will be based.

The Required flag is used for creating tables for determining required fields.

The SourceTable and SourceField properties have the **Read-Only** attribute.

See Also	Fields Object, Index Object, Property Object, Properties Object, Relation Object, TableDef Object, CreateField Method

Indexes Object

Indexes Object

The **Indexes** object represents the collection of the Index objects and provides methods for controlling and accessing the stored objects.

Methods

Count	Returns the number of objects, stored in the collection.
Append	Adds a new object to the collection.
GetByName	Gets an object from the collection by its name.

GetByNumber	Gets an object from the collection by its number.
DeleteByName	Removes an object from the collection by its name.
DeleteByNum	Removes an object from the collection by its number.
Refresh	Refreshes the object collection.

Remarks

The **Indexes** object belongs to the TableDef object and can be retrieved by calling the Indexes method.

See Also Index Object, TableDef Object, Indexes Method

Index Object

Index Object

Represents an index of a database table. An instance of this object can be retrived by using the CreateIndex method of the TableDef object, or from the Indexes collection.

Properties

Name	The name of the object for indentification in the collection.
Foreign	A flag, indicating whether the index is an external key.
Primary	A flag, indicating whether the index is a primary key.
Unique	A flag, indicating whether the given index is unique within the scope of the table, that contains it.

Methods

CreateField	Defines an existing table field, on which the given index will be based. Returns a Field object.
CreateProperty	Creates a Property object, that describes a user-defined property.
Fields	Provides access to the Fields collection.

Properties	Provides access to the Properties collection.

Remarks

New fields will be associated with the given index only after the Append method of the Fields collection has been called.

See Also Field Object, Fields Object, Indexes Object, Property Object, Properties Object, TableDef Object, CreateIndex Method

Parameters Object

Parameters Object

The **Parameters** object represents a collection of the Parameter objects and provides methods for controlling and accessing the stored objects.

Methods

Count	Returns the number of objects, stored in the collection.
Append	Adds a new object to the collection.
GetByName	Gets an object from the collection by its name.
GetByNumber	Gets an object from the collection by its number.
DeleteByName	Removes an object from the collection by its name.
DeleteByNum	Removes an object from the collection by its number.
Refresh	Refreshes the object collection.

Remarks

The **Parameters** object belongs to the QueryDef object and can be retrieved by calling the Parameters method.

For existing stored procedures the call of the Append, DeleteByName and DeleteByNum methods does not cause an actual change of the number of parameters.

See Also Parameter Object, QueryDef Object, Parameters Method

Parameter Object

Parameter Object

The **Parameter** object represents a parameter of stored procedure. An instance of this object can be retrieved by using the CreateParameter method or from the Parameters collection.

Properties

Name	The name of the object for indentification in the collection.
Description	The parameter description.
Type	The parameter type.
Size	The parameter size.
Scale	The number of digits after the decimal point.
AsString	Represents the contents of the parameter as a string.
AsLong	Represents the contents of the parameter as an integer number.
AsDouble	Represents the contents of the parameter as an real number.
AsBoolean	Represents the contents of the parameter as a boolean value.

Methods

CreateProperty	Creates a Property object, that describes a user-defined property.
Properties	Provides access to the Parameters collection.

Remarks

Property objects can be built-in or user-defined.

To create a user-defined **Property** object for one of the objects, call the CreateProperty method of this object, and then add it to the Properties collection with the Append method. Only user-defined objects can be removed from teh Properties collection.

See Also Properties Object, Append Method, CreateProperty Method

Properties Object

Properties Object

The **Properties** object represents a collection of the Property objects and provides methods for controlling and accessing the stored objects.

Methods

Count	Returns the number of objects, stored in the collection.
Append	Adds a new object to the collection.
GetByName	Gets an object from the collection by its name.
GetByNumber	Gets an object from the collection by its number.
DeleteByName	Removes an object from the collection by its name.
DeleteByNum	Removes an object from the collection by its number.

Remarks

The **Properties** object belongs to the Field and Index objects and can be retrieved by calling the Properties method.

See Also Field Object, Index Object, Property Object, Properties Method

Property Object

Property Object

The **Property** object represents a certain property of an object. An instance of the**Property** object can be retrieved by using the CreateProperty method for all objects except collections, or from the Properties collection.

Properties

Name	The name of the object for indentification in the collection.
AsString	Represents the contents of the property as a string.
AsLong	Represents the contents of the property as an integer number.
AsDouble	Represents the contents of the property as an real number.
AsBoolean	Represents the contents of the property as a boolean value.

Remarks

Property objects can be built-in or user-defined.

To create a user-defined **Property** object for one of the objects, call the CreateProperty method of this object, and then add it to the Properties collection with the Append method. Only user-defined objects can be removed from teh Properties collection.

See Also Properties Object, Append Method, CreateProperty Method

QueryDefs Object

QueryDefs Object

The **QueryDefs** object represents a collection of the QueryDef objects and provides methods for controlling and accessing the stored objects.

Methods

Count	Returns the number of objects, stored in the collection.

Append	Adds a new object to the collection.
GetByName	Gets an object from the collection by its name.
GetByNumber	Gets an object from the collection by its number.
DeleteByName	Removes an object from the collection by its name.
DeleteByNum	Removes an object from the collection by its number.

Remarks

The **QueryDefs** object belongs to the Connection and Database objects and can be retrieved by calling the QueryDefs method.

See Also Connection Object, Database Object, QueryDef Object, QueryDefs Method

QueryDef Object

QueryDef Object

Represents a stored SQL procedure. An instance of this object can be retrieved with the CreateQueryDef method of the Connection and Database objects, or from the QueryDefs collection.

Properties

Name	The name of the object for indentification in the collection.
SQL	The stored SQL query.

Methods

CreateParameter	Creates a Parameter object, that describes a parameter of stored procedure.
Parameters	Provides access to the Parameters collection.
CreateProperty	Creates a Property object, that describes a user-defined property.
Properties	Provides access to the Properties collection.

Close	Stops using the object and removes it from the collection.

Remarks

You can create a **QueryDef** object by using the CreateQueryDef method of the Connection and Database objects. The stored variable is created in the database when calling the Append method of the QueryDefs collection.

An SQL query stored in the **QueryDef** object can be executed with the help of the OpenRecordset method of the Connection and Database objects, with the name of the **QueryDef** object as the parameter.

QueryDef objects are based on the **Transact-SQL** standard. If the server doesn't support this standard, using **QueryDef** will cause an error. In this case you can use stored procedures only with the help of direct queries, executed by the OpenRecordset method.

See Also	Connection Object, Database Object, Property Object, Properties Object, QueryDefs Object, Append Method, CreateQueryDef Method, OpenRecordset Method.

Recordsets Object

Recordsets Object

The **Recordsets** method represents the collection of the Recordset objects and provides methods for controlling and accessing the stored objects.

Methods

Count	Returns the number of objects, stored in the collection.
GetByName	Gets an object from the collection by its name.
GetByNumber	Gets an object from the collection by its number.
DeleteByName	Removes an object from the collection by its name.
DeleteByNum	Removes an object from the collection by its number.

Remarks

The **Recordsets** object belongs to the Connection and Database objects and can be retrieved by calling the Recordsets method.

See Also Connection Object, Database Object, Recordset Object, Recordsets Method

Recordset Object

Recordset Object

Represents the result of an SQL query to a database or executing a stored procedure. Provides methods for viewing and modifying the resulting data. An instance of this object can be retrieved by using the OpenRecordset method of the Connection, Database, TableDef objects or from the Recordsets collection.

Properties

Name	The name of the object for indentification in the collection.
BOF	The sign that the beginning of the record has been reached.
EOF	The sign that the end of the record has been reached.

Methods

GetRows	Gets the number of records in the resulting multitude.
AddNew	Adds a new empty line to the open table. Returns a **TRUE** or **FALSE** value.
Delete	Removes a line from the table.
Update	Updates the read record in the table in accordance with the the state of the object's record. Returns a **TRUE** or **FALSE** value.
RowsAffected	The number of records, affected by the last operation.
CanMove	The method determines, whether the pointer can move in the record within the specified direction. Returns a **TRUE** or **FALSE** value.
Move	Moves the pointer within the record in the desired direction. Returns a **TRUE** or **FALSE** value.
MoveFirst	Moves the pointer to the first line of the record. Returns a **TRUE** or **FALSE** value.
MoveLast	Moves the pointer to the last line of the record. Returns a **TRUE** or **FALSE** value.

MoveNext	Moves the pointer to the next line of the record. Returns a **TRUE** or **FALSE** value.
MovePrevious	Moves the pointer to the previous line of the record. Returns a **TRUE** or **FALSE** value.
MoreResults	Checks whether there are more results of the query, updates the resulting multitude and moves the pointer to the starting position. Returns a **TRUE** or **FALSE** value.
CreateProperty	Creates a Property object, that describes a user-defined property.
Fields	Provides access to the Fields collection.
Properties	Provides access to the Properties collection.
Close	Closes the **Recordset** object and releases data, related to it.

Remarks

In some cases it's not possible to determine the number of records in the resulting multitude. The majority of data sources don't let determine the size of the resulting multitude when performing the **SELECT** operation, but determine the size successfully after **UPDATE**, **DELETE**, **INSERT**. If the number of strings is unknown, the GetRows method returns **-1**.

If the given object is not a table record, the AddNew, Delete and Update methods won't perform any action and will return **FALSE** .

The **RowsAffected** method returns the number of records, affected by the AddNew, Delete and Update methods. If the number of strings is unknown, the method returns **-1**.

The ability of the pointer to move in this or that direction is determined by its type, set in the OpenRecordset method, and the level of support of this functionality by the driver and the database. Use the CanMove method to find this out.

See Also Connection Object, Database Object, Fields Object, Property Object, Properties Object, Recordsets Object, TableDef Object, OpenRecordset Method

Relations Object

Relations Object

The **Relations** object represents the collection of the Relation objects, and provides methods for controlling and accessing the stored objects.

Methods

Count	Returns the number of objects, stored in the collection.
Append	Adds a new object to the collection.
GetByName	Gets an object from the collection by its name.
GetByNumber	Gets an object from the collection by its number.
DeleteByName	Removes an object from the collection by its name.
DeleteByNum	Removes an object from the collection by its number.
Refresh	Refreshes the object collection.

Remarks

The **Relations** object belongs to the Database object and can be retrieved by calling the Relations method.

See Also Database Object, Relation Object, Relations Method

Relation Object

Relation Object

Describes relations between the fields of the table. An instance of this object can be retrieved with the help of the CreateRelation method of the Database object or from the Relations collection.

Properties

Name	The name of the object for indentification in the collection.
Table	The name of the table that contains the primary key.
ForeignTable	The name of the table, to create relation with.

Methods

CreateField	Defines the Field object, on which the given relation will be based.
CreateProperty	Creates a Property object, that describes a user-defined property.
Fields	Provides access to the Fields collection.
Properties	Provides access to the Properties collection.

Remarks

The **Relation** object can be based only on the fields of the table, specified in Table, relating to the primary field of this table.

For all created Field objects, the ForeignName property must be defined - it represents the name of the field in the table, being related, on which the external key will be based. The number of fields in the primary and external keys must be the same - i.e., all created fields must have different names and ForeignName properties.

The created fields are added to the Fields collection with the Append method.

The **Relation** object is created in the database on calling the Append method of the Relations collection.

See Also	Database Object, Field Object, Fields Object, Property Object, Properties Object, Relations Object, Append Method, CreateRelation Method, ForeignName Property.

TableDefs Object

TableDefs Object

The **TableDefs** object represents the collection of the TableDef objects and provides methods for controlling and accessing stored objects.

Methods

Count	Returns the number of objects, stored in the collection.
Append	Adds a new object to the collection.
GetByName	Gets an object from the collection by its name.

GetByNumber	Gets an object from the collection by its number.
DeleteByName	Removes an object from the collection by its name.
DeleteByNum	Removes an object from the collection by its number.
Refresh	Refreshes the object collection.

Remarks

The **TableDefs** object belongs to the Database object and can be retrieved by calling the TableDefs method.

See Also Database Object, TableDef Object, TableDefs Method

TableDef Object

TableDef Object

Represents an object, describing a database. Controls existing fields and indexes of the table and allows to create new ones. An instance of this object can be retrieved by using the CreateTableDef method of the Database object, or from the TableDefs collection.

Properties

Name	The name of the object for indentification in the collection.

Methods

CreateField	Creates a new field in the table. Returns a Field object.
CreateIndex	Creates a new index in the table. Returns an Index object.
OpenRecordset	Reads the entire contents of the table. Returns a Recordset object.
CreateProperty	Creates a Property object, that describes a user-defined property.
Fields	Provides access to the Fields collection.
Indexes	Provides access to the Indexes collection.

Properties	Provides access to the Properties collection.

Remarks

New fields and indexes will be added to the table only on calling the Append method of the Fields and Indexes collections respectively.

See Also	Database Object, Field Object, Fields Object, Index Object, Indexes Object, Property Object, Properties Object, Recordset Object, TableDefs Object, Append Method, CreateTableDef.

Workspaces Object

Workspaces Object

The **Workspaces** object represents the collection of the Workspace objects and provides methods for controlling and accessing the stored objects.

Methods

Count	Returns the number of objects, stored in the collection.
GetByName	Gets an object from the collection by its name.
GetByNumber	Gets an object from the collection by its number.
DeleteByName	Removes an object from the collection by its name.
DeleteByNum	Removes an object from the collection by its number.

Remarks

The **Workspaces** object belongs to the DBEngine object and can be retrieved by calling the Workspaces method.

See Also	DBEngine Object, Workspace Object, Workspaces Method

Workspace Object

Workspace Object

Controls connections with databases and transactions. An instance of this object can be retrieved by using the CreateWorkspace method of the DBEngine object, or from the Workspaces collection.

Properties

Name	The name of the object for indentification in the collection.
UserName	The name of the user for accessing the database.
Password	The password for accessing the database.
LoginTimeout	Database login timeout.
QueryTimeout	Query timeout.

Methods

OpenConnection	Creates a database connection. Returns a Connection object.
OpenDatabase	Creates a connection and projects the model of the database. Returns a Database object.
CreateProperty	Creates a Property object, that describes a user-defined property.
Connections	Provides access to the Connections collection.
Databases	Provides access to the Databases collection.
Properties	Provides access to the Properties collection.
BeginTrans	Begins a transaction. All subsequent actions on the database will form Element of this transaction.
CommitTrans	Applies all database changes since BeginTrans was called.
RollbackTrans	Ignores all changes in the database, occurred since BeginTrans was called. The database will be in the same state as before calling BeginTrans.
Close	Closes all open database connections.

Remarks

The UserName and Password properties will be used for authentification in the OpenConnection and OpenDatabase methods, if the corresponding connection parameters are omitted.

By default, the LoginTimeout and QueryTimeout properties set a 15 second interval.

Using transactions at this level assumes that transactions will be used for all Connection and Database objects, that belong to the given **Workspace** object. Transactions can be also controlled at the level of the Connection and Database objects.

If a transaction hasn't been closed before calling the Close method, the CommitTrans function will be called automatically for the changes to come into force.

See Also	Connection Object, Connections Object, Database Object, Databases Object, DBEngine Object, Property Object, Properties Object, Workspaces Object, CreateWorkspace Method.

Databases access Objects Properties

AsBoolean Property

AsBoolean Property

The **AsBoolean** property provides access to the object as to a **Boolean** variable.

Applies to objects: Field, Property.

Syntax

[[Let] *RetVal* = **]** *object*.**AsBoolean**

[Let] *object*.**AsBoolean** = *SetVal*

The **AsBoolean** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable.
SetVal	Required. A **Boolean** type variable.

Remarks

There are also the AsDouble, AsLong and AsString properties, which treat an object as a **Double**, **Long** or **String** variable respectively.

Example

. . . .

See Also Field Object, Property Object, AsDouble Property, AsLong Property, AsString Property.

AsDouble Property

AsDouble Property

The **AsDouble** property provides access to the object as to a **Double** variable.

Applies to objects: Field, Property.

Syntax

[[Let] *RetVal* = **]** *object*.**AsDouble**

[Let] *object*.**AsDouble** = *SetVal*

The **AsDouble** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Double** type variable.
SetVal	Required. A **Double** type variable.

Remarks

There are also the AsBoolean, AsLong and AsString properties, which treat an object as a **Boolean**, **Long** or **String** variable respectively.

Example

. . . .

See Also Field Object, Property Object, AsBoolean Property, AsLong Property, AsString Property.

AsLong Property

AsLong Property

The **AsLong** property provides access to the object as to a **Long** variable.

Applies to objects: Field, Property.

Syntax

[[Let] *RetVal* = **]** *object*.**AsLong**

[Let] *object*.**AsLong** = *SetVal*

The **AsLong** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Long** type variable.
SetVal	Required. A **Long** type variable.

Remarks

There are also the AsBoolean, AsDouble and AsString properties, which treat an object as a **Boolean**, **Double** or **String** variable respectively.

Example

. . . .

See Also	Field Object, Property Object, AsBoolean Property, AsDouble Property, AsString Property.

AsString Property

AsString Property

The **AsString** property provides access to the object as to a **String** variable.

Applies to objects: Field, Property.

Syntax

[[Let] *RetVal* = **]** *object*.**AsString**

[Let] *object*.**AsString** = *SetVal*

The **AsString** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **String** type variable.
SetVal	Required. A **String** type variable.

Remarks

There are also the AsBoolean, AsDouble and AsLong properties, which treat an object as a **Boolean**, **Double** or **Long** variable respectively.

Example

. . . .

See Also Field Object, Property Object, AsBoolean Property, AsDouble Property, AsLong Property.

BOF Property

BOF Property

The **BOF** property indicates that the pointer has reached the beginning of the record and can't be moved any more backwards.

Applies to objects: Recordset.

Syntax

[[Let] *RetVal* = **]** *object*.**BOF**

The **BOF** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable.

Remarks

This property is **Read-Only**.

Example

. . . .

See Also Recordset Object, EOF Property.

ConformanceLevel Property

ConformanceLevel Property

The **ConformanceLevel** property specifies the functionality level of the driver.

Applies to objects: Connection, Database.

Syntax

[[Let] *RetVal* = **]** *object*.**ConformanceLevel**

The **ConformanceLevel** property syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Long** type variable.

Remarks

The **ConformanceLevel** property can have the following values:
cdbLevel0 - supports the minimum SQL level.
cdbLevel1 - supports the basic SQL level.
cdbLevel2 - supports the advanced SQL level.

This property is **Read-Only**.

Example

. . . .

See Also Connection Object, Database Object, DriverVersion Property.

CursorTypes Property

CursorTypes Property

The **CursorTypes** property is a bit mask, that indicates supported cursor types.

Applies to objects: Connection, Database, TableDef.

Syntax
[[Let] *RetVal* = **]** *object*.**CursorTypes**

The **CursorTypes** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Long** type variable.

Remarks

The **CursorTypes** property can contain a combination of the following values:
cdbCSForwardOnly - supports the **cdbCursorForwardOnly** cursor type.
cdbCSKeySet - supports the **cdbCursorKeySet** cursor type.
cdbCSDynamic - cursor type **cdbCursorDynamic** cursor type.
cdbCSStatic - cursor type **cdbCursorStatic** cursor type.

This property is **Read-Only**.

Example
. . . . .

See Also	Connection Object, Database Object, TableDef Object, OpenRecordset Method.

Description Property

Description Property

The **Description** property indicates the purpose of stored procedure parameter.

Applies to objects: Parameter

Syntax

[[Let] *RetVal* = **]** *object*.**Description**

[Let] *object*.**Description** = *SetVal*

The **Name** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. An **Integer** type variable.
SetVal	Required. An **Integer** type variable.

Remarks

The description property of an object can possess the following values (see Parameter Types).

This property changes will be accepted only during parameter creation. In any other cases the changes will be ignored.

Example

. . . .

See Also Parameter Object, Constants.

DriverManager Property

DriverManager Property

The **DriverManager** property contains the name of the used ODBC library.

Applies to objects: DBEngine.

Syntax

[[Let] *RetVal* = **]** *object*.**DriverManager**

[Let] *object*.**DriverManager** = *SetVal*

The **DriverManager** property syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **String** type variable.
SetVal	Required. A **String** type variable.

Remarks

By default, the **DriverManager** property has the **odbc32.dll** value for Windows and **iODBC CFM Bridge** for Mac OS.

This property must be modified before opening a database connection.

Example

. . . .

See Also DBEngine Object.

DriverType Property

DriverType Property

The **DriverType** property stores the type of the used driver.

Applies to objects: DBEngine.

Syntax

[[Let] *RetVal* = **]** *object*.**DriverType**

[Let] *object*.**DriverType** = *SetVal*

The **DriverType** property syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Long** type variable.
SetVal	Required. A **Long** type variable.

Remarks

Presently, only ODBC drivers are fully supported. The **DriverType** property is reserved for future use. The changes you make in this property are ignored.

Example

. . . .

See Also DBEngine Object.

DriverVersion Property

DriverVersion Property

The **DriverVersion** property specifies the version of the used ODBC driver.

Applies to objects: Connection, Database.

Syntax

[[Let] *RetVal* = **]** *object*.**DriverVersion**

The **DriverVersion** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Double** type variable.

Remarks

This property is **Read-Only**.

Example

. . . .

See Also Connection Object, Database Object, ConformanceLevel Property.

EOF Property

EOF Property

The **EOF** property indicates that the pointer has reached the end of the record and can't be moved forward any further.

Applies to objects: Recordset.

Syntax

[[Let] *RetVal* = **]** *object*.**EOF**

The **BOF** property syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable.

Remarks

This property is **Read-Only**.

Example

. . . .

See Also Recordset Object, BOF Property.

ForeignName Property

ForeignName Property

The **ForeignName** stores the name of the field of the table, referenced by the Relation table, on which the external key will be based.

Applies to objects: Field.

Syntax

[[Let] *RetVal* = **]** *object*.**ForeignName**

[Let] *object*.**ForeignName** = *SetVal*

The **ForeignName** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **String** type variable.
SetVal	Required. A **String** type variable.

Remarks

The **ForeignName** must be necessarily set to all Field objects, located in the Fields collection of the Relation object.

All **ForeignName** properties must have different values. The fields, specified in **ForeignName** must exist in the referenced table.

Example

. . . .

See Also Field Object, Fields Object, Relation Object, CreateRelation Method.

ForeignTable Property

ForeignTable Property

The **ForeignTable** property stores the name of the table, in which the external key is to be created.

Applies to objects: Relation.

Syntax

[[Let] *RetVal* = **]** *object*.**ForeignTable**

[Let] *object*.**ForeignTable** = *SetVal*

The **ForeignTable** property syntax has these Elements:

Element	**Description**

object	Required. A reference to an instance of the object.
RetVal	Optional. A **String** type variable.
SetVal	Required. A **String** type variable.

Remarks

On creating a Relation object, an external key will be created in the table, referenced by the **ForeignTable** property. This key will be based on the fields of the table, specified in the ForeignName properties of the Field objects that make Element of the Fields collection of the Relation object.

The number of fields in the primary and external keys of the referenced table must coincide, that is, all created fields must have different names and ForeignName properties.

Example

. . . .

See Also Relation Object, Field Object, Fields Object, ForeignName Property, Table Property, CreateRelation Method.

IsolationLevels Property

IsolationLevels Property

The **IsolationLevels** property is a bit mask, that indicates the level of transaction support.

Applies to objects: Connection, Database.

Syntax

[[Let] *RetVal* = **]** *object*.**IsolationLevels**

The **Transactions** property syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Long** type variable.

Remarks

The following situations may occur when using transactions:

Dirty Read. Transaction 1 changes a row. Transaction 2 reads the changed row before transaction 1 commits the change. If transaction 1 rolls back the change, transaction 2 will have read a row that is considered to have never existed.
Nonrepeatable Read. Transaction 1 reads a row. Transaction 2 updates or deletes that row and commits this change. If transaction 1 attempts to reread the row, it will receive different row values or discover that the row has been deleted.
Phantom. Transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 generates one or more rows (through either inserts or updates) that match the search criteria. If transaction 1 reexecutes the statement that reads the rows, it receives a different set of rows.

The **IsolationLevels** property may contain a combination of the following values:
cdbTransReadUncommited - Dirty reads, nonrepeatable reads, and phantoms are possible.
cdbTransReadCommited - Dirty reads are not possible. Nonrepeatable reads and phantoms are possible.
cdbTransRepeatableRead - Dirty reads and nonrepeatable reads are not possible. Phantoms are possible.
cdbTransSerializable - Transactions are serializable. Serializable transactions do not allow dirty reads, nonrepeatable reads, or phantoms.

If equal to **0** - transactions are not supported.

This property is **Read-Only**.

Example
. . . . .

See Also	Connection Object, Database Object, Transactions Property, BeginTrans Method.

LoginTimeout Property

LoginTimeout Property

The **LoginTimeout** specifies the maximum database connection timeout delay.

Applies to objects: Workspace.

Syntax
[[Let] *RetVal* = **]** *object*.**LoginTimeout**

[Let] *object*.**LoginTimeout** = *SetVal*

The **LoginTimeout** property syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Long** type variable.
SetVal	Required. A **Long** type variable.

Remarks

The default timeout delay is 15 seconds.

Example

. . . .

See Also Workspace Object, QueryTimeout Property.

Name Property

Name Property

The **Name** property stores the name of the object for identifying in the database and object collection.

Applies to objects: Connection, Database, Field, Index, Property, QueryDef, Recordset, Relation, TableDef, Workspace.

Syntax

[[Let] *RetVal* = **]** *object*.**Name**

[Let] *object*.**Name** = *SetVal*

The **Name** property syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **String** type variable.
SetVal	Required. A **String** type variable.

Remarks

The name property of an object must be unique within the scope of the collection.

Example

. . . .

See Also	Connection Object, Database Object, Field Object, Index Object, Property Object, QueryDef Object, Recordset Object, Relation Object, TableDef Object, Workspace Object.

Password Property

Password Property

The **Password** property stores the user password for accessing the database.

Applies to objects: Workspace.

Syntax

[[Let] *RetVal* = **]** *object*.**Password**

[Let] *object*.**Password** = *SetVal*

The **Password** property syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **String** type variable.
SetVal	Required. A **String** type variable.

Remarks

The UserName and **Password** properties will be used for all database connections within the scope of the given Workspace object, unless the initialization string of the connection is not specified in the OpenConnection and OpenDatabase methods, or the **UID** and **PWD** indetifiers are missing in it.

Example

. . . .

See Also	Workspace Object, UserName Property, OpenConnection Method, OpenDatabase Method.

Primary Property

Primary Property

The **Primary** property indicates whether the given index is a primary key.

Applies to objects: Index.

Syntax

[[Let] *RetVal* = **]** *object*.**Primary**

[Let] *object*.**Primary** = *SetVal*

The **Primary** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable.
SetVal	Required. A **Boolean** type variable.

Remarks

This property can only be defined when index is created. Otherwise modifying this property is ignored.

A table may have only one primary key.

Example

. . . .

See Also	Index Object, TableDef Object, Unique Property.

QueryTimeout Property

The **QueryTimeout** sets the maximum timeout for the operation.

Applies to objects: Workspace.

Syntax

[[Let] *RetVal* = **]** *object*.**QueryTimeout**

[Let] *object*.**QueryTimeout** = *SetVal*

The **QueryTimeout** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Long** type variable.
SetVal	Required. A **Long** type variable.

Remarks

The default timeout delay is 15 seconds.

Example

. . . .

See Also Workspace Object, LoginTimeout Property.

Required Property

The **Required** property specifies, whether the given field is required.

Applies to objects: Field.

Syntax

[[Let] *RetVal* = **]** *object*.**Required**

[Let] *object*.**Required** = *SetVal*

The **Required** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable.
SetVal	Required. A **Boolean** type variable.

Remarks

This property can be set only when creating tables, for determining required fields. Otherwise modifying this property is ignored.

Example

. . . .

See Also Field Object.

Scale Property

Scale Property

The **Scale** property stores object size.

Applies to objects: Field, Parameter

Syntax

[[Let] *RetVal* = **]** *object*.**Scale**

[Let] *object*.**Scale** = *SetVal*

The **Scale** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. An **Integer** type variable.
SetVal	Required. An **Integer** type variable.

Remarks

This property can be used just for **cddbCurrency**, **cddbNumeric** and **cddbDecimal** types (see Data Types).

This property changes will be accepted only during parameter creation. In any other cases the changes will be ignored.

Example

. . . .

See Also Field Object, Parameter Object, Size Property, Constants.

Size Property

Size Property

The **Size** property stores object size.

Applies to objects: Field, Parameter

Syntax

[[Let] *RetVal* = **]** *object*.**Size**

[Let] *object*.**Size** = *SetVal*

The **Size** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Long** type variable.
SetVal	Required. A **Long** type variable.

Remarks

This property can be used just for **cddbText**, **cddbBinary**, **cddbMemo**, **cddbCurrency**, **cddbNumeric** and **cddbDecimal** types (see Data Types).

This property changes will be accepted only during parameter creation. In any other cases the changes will be ignored.

Example

. . . .

See Also Field Object, Parameter Object, Constants.

SourceField Property

SourceField Property

The **SourceField** property stores the name of the field as it's represented in the database.

Applies to objects: Field.

Syntax

[[Let] *RetVal* = **]** *object*.**SourceField**

The **SourceField** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **String** type variable.

Remarks

This property is **Read-Only**.

Example

. . . .

See Also Field Object, SourceTable Property.

SourceTable Property

SourceTable Property

The **SourceTable** property contains the name of the table, that owns the given field, as it's represented in the database.

Applies to objects: Field.

Syntax

[[Let] *RetVal* = **]** *object*.**SourceTable**

The **SourceTable** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **String** type variable.

Remarks

This property is **Read-Only**.

Example

. . . .

See Also Field Object, SourceField Property.

SQL Property

SQL Property

The **SQL** property stores the SQL code of the stored procedure.

Applies to objects: QueryDef.

Syntax

[[Let] *RetVal* = **]** *object*.**SQL**

[Let] *object*.**SQL** = *SetVal*

The **SQL** property syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **String** type variable.
SetVal	Required. A **String** type variable.

Remarks

You can execute the given SQL query by using the OpenRecordset method, with the QueryDef object name as the parameter.

Example

. . . .

See Also QueryDef Object, OpenRecordset Method.

Table Property

Table Property

The **Table** property stores the name of the table that contains the primary key.

Applies to objects: Relation.

Syntax

[[Let] *RetVal* = **]** *object*.**Table**

[Let] *object*.**Table** = *SetVal*

The **Table** property syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **String** type variable.
SetVal	Required. A **String** type variable.

Remarks

The Relation object can be based only on the fields of the table, specified in the **Table** property and relating to the primary key of that table.

You need to create as many fields for the given Relation object, as there are fields relating to the primary key, specified in the **Table** property.

Example

. . . .

See Also	Relation Object, Field Object, ForeignName Property, ForeignTable Property, CreateField Method, CreateRelation Method.

Transactions Property

Transactions Property

The **Transactions** indicates transaction support.

Applies to objects: Connection, Database.

Syntax

[Let] *RetVal* = *object*.**Transactions**

The **Transactions** property syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable.

Remarks

This property is **Read-Only**.

Example

```
Dim en As dbEngine
Set en = new dbEngine
Dim ws As Workspace
Set ws = en.CreateWorkspace("IBWorkspace")
Dim db As Database
Set db = ws.OpenDatabase("IBBase", FALSE, FALSE, "ODBC; UID=SYSDBA;
PWD=masterkey; DSN=IBBase")
If db.Transactions Then
 db.BeginTrans()
```

```
  ...
  db.CommitTrans()
End IF
db.Close()
ws.Close()
```

See Also	Connection Object, Database Object, IsolationLevels Property, BeginTrans Method.

Type Property

Type Property

The **Type** property represents object type.

Applies to objects: Field, Parameter

Syntax

[[Let] *RetVal* = **]** *object*.**Type**

[Let] *object*.**Type** = *SetVal*

The **Type** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. An **Integer** type variable.
SetVal	Required. An **Integer** type variable.

Remarks

The description property of an object can possess the following values (see Data Types).

This property changes will be accepted only during parameter creation. In any other cases the changes will be ignored.

Example

```
.......
```

See Also Field, Parameter Object, Constants.

Unique Property

Unique Property

The **Unique** property determines, whether the given index is unique.

Applies to objects: Index.

Syntax

[[Let] *RetVal* = **]** *object*.**Unique**

[Let] *object*.**Unique** = *SetVal*

The **Unique** property syntax has these Elements:

Element	**Description**
object	Required. Link to the object instance
RetVal	Optional. A variable of **Boolean** type.
SetVal	Required. A variable of **Boolean** type.

Remarks

This property can be set only when creating the index. Otherwise modifying this property is ignored.

If the index is unique, it means that the fields it's based on, are different. This guarantees that all operations that use this index will affect exactly the chosen records. Besides, indexing decreases the record search time.

Example

. . . .

See Also Index Object, Pimary Property.

UserName Property

The **UserName** property stores the user name for accessing the database.

Applies to objects: Workspace.

Syntax

[[Let] *RetVal* = **]** *object*.**UserName**

[Let] *object*.**UserName** = *SetVal*

The **UserName** property syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **String** type variable.
SetVal	Required. A **String** type variable.

Remarks

The **UserName** and Password properties will be used for all database connections within the scope of the given Workspace object, unless the initialization string of the connection is not specified in the OpenConnection and OpenDatabase methods, or the **UID** and **PWD** indetifiers are missing in it.

Example

. . . .

See Also	Workspace Object, Password Property, OpenConnection Method, OpenDatabase Method.

Databases access Objects Methods

AddNew Method

AddNew Method

Adds a new empty string to the open table.

Applies to objects: Recordset.

Syntax
[[**Let**] *RetVal* =] *object*.**AddNew**()

The **MethodName** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable, that indicates whether the string was added.

Remarks

If the given object is not a table record, the method won't perform any action and will return **FALSE**.

Example
. . . .

See Also Recordset Object, Delete Method.

Append Method

Append Method

Appends an object to the collection.

Applies to objects: Connections, Databases, Fields, Indexes, Properties, QueryDefs, Recordsets, Relations, TableDefs, Workspaces.

Syntax
object.**Append**(*addObj*)

The **Append** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
addObj	Required. An instance of the object to be added to the collection.

Remarks

The **Append** lets append to the collection an object with a unique Name property. If such object already exists in the collection, the new object won't be added.

The **Append** method should be called after creating the Field, Index, Property, QueryDef, Recordset, Relation, TableDef object with the CreateField, CreateIndex, CreateProperty, CreateQueryDef, OpenRecordset, CreateRelation, CreateTableDef methods respectively and after assigning all necessary properties to the created objects. After calling the **Append** method the corresponding objects (except Property) will appear in the database.

Example
```
Examples for most common cases.
```

See Also	Connections Object, Databases Object, Field Object, Fields Object, Index Object, Indexes Object, Property Object, Properties Object, QueryDef Object, QueryDefs Object, Recordset Object, Recordsets Object, Relation Object, Relations Object, TableDef Object, TableDefs Object, Workspaces Object, Name Property, CreateField Method, CreateIndex Method, CreateProperty Method, CreateQueryDef Method, CreateRelation Method, CreateTableDef Method, OpenRecordset Method.

BeginTrans Method

BeginTrans Method

Begins a transaction.

Applies to objects: Connection, Database, Workspace.

Syntax
[[**Let**] *RetVal* =] *object*.**BeginTrans**([*IsolationLevel*])

The **BeginTrans** method syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable that indicates whether the transaction could be started.
IsolationLevel	Optional. A **Long** type variable, indicating the isolation level of the transaction.

Remarks

The **BeginTrans** method switches the transaction mechanism from automatical to manual execution mode. In the automatical mode it's not possible to organize all database changes into one block and is not possible to cancel transactions. In the manual execution mode all operations following **BeginTrans** are treated as a single block, executed with the CommitTrans, and cancelation of a transaction is handles by the RollbackTrans function.

The Connection and Database allow to define the transaction support and available transaction isolation levels for the database connection described by them with the help of the Transactions and IsolationLevels methods. The **BeginTrans** method, that belongs to these objects, can take the *IsolationLevel* parameter, that sets the isolation level for the transaction. This parameter can be omitted - then the default isolation level for the database will be accepted.

The Workspace object provides general control over transactions. On calling its method **BeginTrans**, the **BeginTrans** of all Connection and Database objects, that belong to it and support transactions, will be called without parameters. The **BeginTrans** method of the Workspace object returns no result.

After calling the CommitTrans and RollbackTrans functions automatical transaction execution mode is restored. To form the next transaction, the **BeginTrans** method must be called again.

If none of these functions is called, transaction will be committed by the CommitTrans function on destroying or closing the object, controlling this transaction (Connection, Database, Workspace).

Example

. . . .

See Also	Connection Object, Database Object, Workspace Object, Transactions Property, IsolationLevels Property, CommitTrans Method, RollbackTrans Method.

CanMove Method

This method determines whether the pointer can be moved in the record in the specified direction.

Applies to objects: Recordset.

Syntax

[[**Let**] *RetVal* =] *object*.**CanMove**(*Direction*)

The **CanMove** method syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable.
Direction	Required. A **Long** type variable, that determines the direction in which the pointer moves.

Remarks

The *Direction* parameter can be a combination of the following values:
cdbMoveNext - the MoveNext function can be used.
cdbMoveFirst - the MoveFirst function can be used.
cdbMoveLast - the MoveLast function can be used.
cdbMovePrevious - the MovePrevious function can be used.
cdbMoveAbsolute - the Move function can be used for absolute positioning.
cdbMoveRelative - the Move function can be used for relative positioning.

The ability of the pointer to move in this or that direction is determined by its type, set in the OpenRecordset method, and by the level of support for this functionality by the driver and the database.

Example

. . . .

See Also Recordset Object, OpenRecordset Method, Move Method, MoveFirst Method, MoveLast Method, MoveNext Method, MovePrevious Method.

Close Method

Close Method

Terminates working with the object.

Applies to objects: Connection, Database, QueryDef, Recordset, Workspace.

Syntax

object.**Close**()

The **MethodName** method syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.

Remarks

For the Recordset object the **Close** method closes the pointer and releases the memory, alloted for the records. For the Connection, Database, Workspace objects in closes the transaction.

For re-using a previously opened object you need to call the OpenConnection, OpenDatabase, OpenRecordset, CreateQueryDef, CreateWorkspace methods.

Example

. . . .

See Also	Connection Object, Database Object, QueryDef Object, Recordset Object, Workspace Object, CreateQueryDef Method, CreateWorkspace Method, OpenConnection Method, OpenDatabase Method, OpenRecordset Method.

CommitTrans Method

CommitTrans Method

Commits a transaction.

Applies to objects: Connection, Database, Workspace.

Syntax

[[**Let**] *RetVal* =] *object*.**CommitTrans**()

The **CommitTrans** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable that indicates whether the transaction has been committed.

Remarks

All operations between BeginTrans and **CommitTrans** will be performed.

On calling the **CommitTrans** method of the Workspace object, **CommitTrans** methods of all its Connection and Database objects will be called. The **CommitTrans** method of the Workspace object returns no result.

On calling the **CommitTrans** method the automatical transaction execution mode is restored. So, to form the next transaction, BeginTrans must be called again.

If transaction is not closed, the **CommitTrans** function will be called on destroying or closing the object, controlling the transaction so that the changes come in force.

Depending on the driver and the database, the **CommitTrans** and RollbackTrans methods can close the open pointers of Recordset objects.

Example

. . . .

See Also Connection Object, Database Object, Workspace Object, BeginTrans Method, RollbackTrans Method.

Connections Method

Connections Method

Provides access to the Connections collection of the Workspace object.

Applies to objects: Workspace.

Syntax
[[**Set**] *RetVal* =] *object*.**Connections**()

The **Connections** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A Connections type variable.

Example
```
' Declare and initialize variables
Dim engine As DBEngine
Set engine = new DBEngine
Dim wspace As Workspaces
Set wspace = engine.CreateWorkspace("MyWorkspace")
' Create connection
Dim conn As Connection
Set conn = wspace.OpenConnection("MyConnection", false, false, "ODBC;
UID=mylogin; PWD=mypassword; DSN=SQLBaseDSN")
'...
' Get the Connections collection
Dim cscoll As Connections
Set cscoll = wspace.Connections()
'...
```

See Also Connection Object, Connections Object, Workspace Object.

Count Method

Count Method

Returns the number of objects, stored in the collection.

Applies to objects: Workspaces, Connections, Databases, TableDefs, QueryDefs, Relations, Indexes, Recordsets, Fields, Properties.

Syntax
[[**Let**] *RetVal* =] *object*.**Count**()

The **Count** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Long** type variable that stores the number of objects in the collection.

Example

```
A simple example.
```

See Also	Connections Object, Databases Object, Fields Object, Indexes Object, Properties Object, QueryDefs Object, Recordsets Object, Relations Object, TableDefs Object, Workspaces Object.

CreateField Method

CreateField Method

Creates a new Field object.

Applies to objects: Index, Relation, TableDef.

Syntax

[[**Set**] *RetVal* =] *object*.**CreateField**([*Name*], [*Type*], [*Size*])

The **MethodName** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A Field type variable.
Name	Optional. A **String** type variable. The name of the field, being created.
Type	Optional. A **Long** type variable. The type of the field.
Size	Optional. A **Long** type variable. The size of the field.

Remarks

The *Type* determines the type of the value to be stored in the field. It can take one of the following values:
cdbBoolean - A boolean variable.
cdbByte - Integer (1 byte).

cdbInteger - Integer (2 bytes).
cdbLong - Integer (4 bytes).
cdbCurrency - Real (can support the currency symbol).
cdbSingle - Real.
cdbDouble - Double-precision real.
cdbDate - Date.
cdbBinary - Binary value.
cdbText - Text.
cdbLongBinary - Binary array.
cdbMemo - Binary array.
cdbGUID - Identifier (4 bytes).
cdbBigInt - Integer (8 bytes).
cdbVarBinary - Binary array.
cdbChar - Character (1 byte and 2 bytes if UNICODE).
cdbNumeric - Real number.
cdbDecimal - Real number.
cdbFloat - Real number.
cdbTime - Time.
cdbTimeStamp - Date and type.
Support for a type depends on the driver and the database.

The values specified in *Type* and *Size* are considered only when creating the table.

The created Field object needs to be added to the Fields collection by using the Append method.

For the Index and Relation objects the *Name* field must exist in the table, on which these objects are based.

Example

. . . .

See Also Field Object, Fields Object, Index Object, Relation Object, TableDef Object, Name Property, Append Method.

CreateIndex Method

CreateIndex Method

Creates a new Index object.

Applies to objects: TableDef.

Syntax

[[**Set**] *RetVal* =] *object*.**CreateIndex**([*Name*])

The **CreateIndex** method syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. An Index type variable.
Name	Optional. A **String** type variable. The name of the index.

Remarks

Once a new Index object has been created, you need to create the fields of the table, on which it will be based. For each such field you need to call the CreateField method with its name and add the created object to the Fields collection by using the Append method.

The index itself will be created in the table on calling the Append method of the Indexes collection.

The index properties Primary and Unique can be defined before calling Append of the Indexes collection. In other cases their changes will be ignored.

Example

. . . .

See Also	Index Object, Indexes Object, TableDef Object, Name Property, Primary Property, Unique Property, Append Method.

CreateParameter Method

CreateParameter Method

Creates a new Parameter, describing a parameter to a stored procedure..

Applies to objects: QueryDef

Syntax

[[**Set**] *RetVal* =] *object*.**CreateParameter**([*Name*], [*Description*], [*Type*], [*Size*], [*Scale*])

The **CreateParameter** method syntax has these Elements:

Element	**Description**
object	Required. Link to object. String variable
RetVal	Optional. Parameter. variable
Name	Optional. **String** variable. Name of new properties.
Description	Optional. **Integer** variable..
Type	Optional. **Integer** variable. Parameter type.
Size	Optional. **Long** variable. Parameter size.
Scale	Optional. **Integer** variable. The number of decimal places.

Remarks

Description variable determines whether the parameter is input (**cddbInput**), output(**cddbOutput**) or both (**cddbInputOutput**) - see.Parameter Types.

The variable *Size* is used only for parameter types : **cddbText**, **cddbBinary**, **cddbMemo**, **cddbCurrency**, **cddbNumeric**, **cddbDecimal** - см. Data Types.

The variable *Scale* is used only for parameter types : **cddbCurrency**, **cddbNumeric**, **cddbDecimal**.

When the Parameter and its properties are created, it should be added to the Parameters collection using the Append method.

Example

. . . .

See Also	Parameter Object, Parameters Object, QueryDef Object, Description Property, Name Property, Size Property, Scale Property, Type Property, Append Method, Constants.

CreateProperty Method

Creates a new Propery object, that describes a user-defined property of an object.

Applies to objects: Connection, Database, DBEngine, Field, Index, QueryDef, Recordset, Relation, TableDef, Workspace.

Syntax

[[**Set**] *RetVal* =] *object*.**CreateProperty**([*Name*], [*Value*], [*Inheritance*])

The **CreateProperty** method syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A Property type variable.
Name	Optional. A **String** type variable. The name of the property being created.
Value	Optional. The value of the property being created.
Inheritance	Optional. A **Boolean** type variable. Defines, whether the property is inherited by child object.

Remarks

If the *Inheritance* flag is set, this property will appear in all child objects of the object.

After creating the property you need to add it to the Properties collection by using the Append method.

Example

. . . .

See Also	Connection Object, Database Object, DBEngine Object, Field Object, Index Object, Propery Object, Properties Object, QueryDef Object, Recordset Object, Relation Object, TableDef Object, Workspace Object, Name Property, Append Method.

CreateQueryDef Method

CreateQueryDef Method

Creates a new QueryDef object.

Applies to objects: Connection, Database

Syntax

[[**Set**] *RetVal* =] *object*.**CreateQueryDef**([*Name*], [*SQLQuery*])

The **CreateQueryDef** method syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A QueryDef type variable.
Name	Optional. A **String** type variable. The name of the object being created.
SQLQuery	Optional. A **String** type variable. An SQL query.

Remarks

The stored procedure will be created in the database after calling the Append method of the QueryDefs collection. It can be performed by the OpenRecordset record, with the name of the QueryDef object as the parameter.

Example

. . . .

See Also Connection Object, Database Object, QueryDef Object, QueryDefs Object, Name Property, SQL Property, Append Method, OpenRecordset Method.

CreateRelation Method

CreateRelation Method

Creates a new Relation object.

Applies to objects: Database

Syntax

[[**Set**] *RetVal* =] *object*.**CreateRelation**([*Name*], [*PrimaryTableName*], [*ForeignTableName*])

The **CreateRelation** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A Relation type variable.
Name	Optional. A **String** type variable. The name of the relation, being created.
PrimaryTableName	Optional. A **String** type variable. The name of the referenced table, that contains the primary key.
ForeignTableName	Optional. A **String** type variable. The name of the referenced table, in which the external key will be created.

Remarks

Once a new Relation object has been created, you need to set the fields of the *PrimaryTableName* table, on which it will be based. For each such field you need to call the CreateField method with its name, set the ForeignName property to it and add the created object to the Fields collection by using the Append method.

The relation itself will be created in the database on calling the Append method of the Relations collection.

Example

. . . .

See Also Database Object, Fields Object, Relation Object, Relations Object, ForeignTable Property, Name Property, Table Property, Append Method, CreateField Method.

CreateTableDef Method

CreateTableDef Method

Creates a new TableDef object.

Applies to objects: Database

Syntax

[[**Set**] *RetVal* =] *object*.**CreateTableDef**([*Name*])

The **CreateTableDef** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A TableDef type variable.
Name	Optional. A **String** type variable. The name of the table being created.

Remarks

Once a new TableDef object has been created, you need to set the fields by using the CreateField method. The created fields are added to the Fields collection by using the Append method.

The table itself will be created in the database on calling the Append method of the TableDefs collection.

Example

. . . .

See Also Database Object, Fields Object, TableDef Object, TableDefs Object, Name Property, Append Method, CreateField Method.

CreateWorkspace Method

CreateWorkspace Method

Creates a Workspace object and adds it to the Workspaces collection of the DBEngine object. Returns a reference to an instance of the Workspace object.

Applies to objects: DBEngine

Syntax

[[**Set**] *RetVal* =] *object*.**CreateWorkspace** ([*workspaceName*], [*userName*], [*password*])

The **CreateWorkspace** method syntax has these Elements:

Element	**Description**

object	Required. A reference to an instance of the object.
workspaceName	Optional. An expression that returns a **String** value. The name of the Workspace object being created. It's used to identify the object in the collection.
userName	Optional. An expression that returns a **String** value. The name of the user.
password	Optional. An expression that returns a **String** value. The password.
RetVal	Optional. A Workspace type variable.

Remarks

The user name and password will be used for connecting to the database unless the connection parameter string is specified in the OpenDatabase or OpenConnection methods of the Workspace object.

The created Workspace object is immediately added to the Workspaces collection of the DBEngine object, and it's not needed to call the Append method for it.

Example

```
Dim engine As dbEngine
Set engine = new dbEngine
' creating new Workspace
Dim wspace As Workspace
Set wspace = engine.CreateWorkspace("MyWorkspace", "mylogin", "mypassword")
'...
```

See Also DBEngine Object, Workspace Object, Workspaces Object.

Databases Method

Databases Method

Provides access to the Databases collection of the Workspace object.

Applies to objects: Workspace

Syntax

[[**Set**] *RetVal* =] *object*.**Databases** ()

The **Databases** method syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A Databases type variable.

Example

```
' Declare and initialize variables
Dim engine As DBEngine
Set engine = new DBEngine
Dim wspace As Workspaces
Set wspace = engine.CreateWorkspace("MyWorkspace")
' Open database
Dim dbase As Database
Set dbase = wspace.OpenDatabase("MyDatabase", 0, false, "ODBC; UID=mylogin;
PWD=mypassword; DSN=SQLBaseDSN")
'...
' Get the  Databases collection
Dim dbcoll As Databases
Set dbcoll = wspace.Databases()
'...
```

See Also Database Object, Databases Object, Workspace Object.

DeleteByName Method

DeleteByName Method

Deletes an object from the collection by its Name property.

Applies to objects: Connections, Databases, Fields, Indexes, Properties, QueryDefs, Recordsets, Relations, TableDefs, Workspaces.

Syntax

object.**DeleteByName**(*Name*)

The **DeleteByName** method syntax has these Elements:

Element	Description
object	Required. An instance of one of the object collections listed above.

Name	Required. An expression that returns a **String** value. The name of the object to be deleted.

Remarks

If the object with the specified *objName* is missing in the collection, the following error occurs: *"The item 'objName' is not found in the collection"*

Example

. . . .

See Also	Connection Object, Connections Object, Database Object, Databases Object, Field Object, Fields Object, Index Object, Indexes Object, Property Object, Properties Object, QueryDef Object, QueryDefs Object, Recordset Object, Recordsets Object, Relation Object, Relations Object, TableDef Object, TableDefs Object, Workspace Object, Workspaces Object, Name Property.

DeleteByNum Method

DeleteByNum Method

Deletes an object from the collection by its index.

Applies to objects: Connections, Databases, Fields, Indexes, Properties, QueryDefs, Recordsets, Relations, TableDefs, Workspaces.

Syntax

object.**DeleteByNum**(*Index*)

The **DeleteByNum** method syntax has these Elements:

Element	Description
object	Required. An instance of one of the object collections listed above.
Index	Required. An expression that returns a **Long** value. The index of the object to be deleted.

Remarks

If the specified *objIndex* index is outside the collection range, the following error occurs: *"The index is out of range"*.

Example

. . . .

See Also Connection Object, Connections Object, Database Object, Databases Object, Field Object, Fields Object, Index Object, Indexes Object, Property Object, Properties Object, QueryDef Object, QueryDefs Object, Recordset Object, Recordsets Object, Relation Object, Relations Object, TableDef Object, TableDefs Object, Workspace Object, Workspaces Object.

Delete Method

Delete Method

The **Delete** method deletes the current string from the table.

Applies to objects: Recordset.

Syntax

[[**Let**] *RetVal* =] *object*.**Delete**()

The **Delete** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable that indicates whether the string could be deleted.

Remarks

If the given object is not a table record, the method won't perform any action and will return **FALSE**.Example

. . . .

See Also Recordset Object, AddNew Method.

Fields Method

Fields Method

Provides access to the Fields collection of the Index, Relation, TableDef objects.

Applies to objects: Index, Relation, TableDef.

Syntax

[[**Set**] *RetVal* =] *object*.**Fields**()

The **Fields** method syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A Fields type variable.

Example

. . . .

See Also Fields Object, Index Object, Relation Object, TableDef Object.

GetByName Method

GetByName Method

Retrieves an object from the collection by its Name property.

Applies to objects: Connections, Databases, Fields, Indexes, Properties, QueryDefs, Recordsets, Relations, TableDefs, Workspaces.

Syntax

[[**Set**] *RetVal* =] *object*.**GetByName**(*Name*)

The **GetByName** method syntax has these Elements:

Element	Description
object	Required. An instance of one of the object collections listed above.

Name	Required. An expression that returns a **String** value. The name of the object to be retrieved.
RetVal	Optional. An instance of an object, retrieved from the collection (of Workspace, Connection, Database, TableDef, QueryDef, Relation, Index, Recordset, Field, Property type respectively).

Remarks

If the object with the specified *objName* is missing in the collection, the following error occurs: *"The item 'objName' is not found in the collection"*.

Example

. . . .

See Also	Connection Object, Connections Object, Database Object, Databases Object, Field Object, Fields Object, Index Object, Indexes Object, Property Object, Properties Object, QueryDef Object, QueryDefs Object, Recordset Object, Recordsets Object, Relation Object, Relations Object, TableDef Object, TableDefs Object, Workspace Object, Workspaces Object, Name Property.

GetByNumber Method

GetByNumber Method

Retrieves an object from the collection by its index.

Applies to objects: Connections, Databases, Fields, Indexes, Properties, QueryDefs, Recordsets, Relations, TableDefs, Workspaces.

Syntax

[[**Set**] *RetVal* =] *object*.**GetByNumber** (*Index*)

The **GetByNumber** method syntax has these Elements:

Element	Description
object	Required. An instance of one of the object collections listed above.
Index	Required. An expression that returns a **Long** value. The index of the object to be retrieved.

RetVal	Optional. An instance of an object retrieved from the collection (of Workspace, Connection, Database, TableDef, QueryDef, Relation, Index, Recordset, Field, Property type respectively).

Remarks

If the specified *objIndex* index is outside the collection range, the following error occurs: *"The index is out of range"*.

Example

. . . .

See Also	Connection Object, Connections Object, Database Object, Databases Object, Field Object, Fields Object, Index Object, Indexes Object, Property Object, Properties Object, QueryDef Object, QueryDefs Object, Recordset Object, Recordsets Object, Relation Object, Relations Object, TableDef Object, TableDefs Object, Workspace Object, Workspaces Object.

GetRows Method

GetRows Method

The **GetRows** method returns the number of records, gotten as the result of an SQL query.

Applies to objects: Recordset.

Syntax

[[**Let**] *RetVal* =] *object*.**GetRows**()

The **GetRows** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Long** type variable, that gets the number of records.

Remarks

In some cases it's not possible to determine the number of records in the resulting multitude. The majority of data sources don't let determine the size of the resulting multitude when performing the **SELECT** operation, but determine the size successfully after **UPDATE**, **DELETE**, **INSERT**. If the number of strings is unknown, the **GetRows** method returns **-1**.

Example

. . . .

See Also Recordset Object, RowsAffected Method.

Indexes Method

Indexes Method

Provides access to the Indexes collection of the TableDef object.

Applies to objects: TableDef.

Syntax

[[**Set**] *RetVal* =] *object*.**Indexes**()

The **Indexes** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. An Indexes type variable.

Example

. . . .

See Also Indexes Object, TableDef Object.

MoveFirst Method

MoveFirst Method

Moves the pointer to the first position upon the result of the query.

Applies to objects: Recordset.

Syntax

[[**Let**] *RetVal* =] *object*.**MoveFirst**()

The **MoveFirst** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable, that indicates whether the pointer was moved to the necessary string.

Remarks

The ability of the pointer to move in such way is determined with the help of the CanMove method with the **cdbMoveFirst** parameter.

Example

. . . .

See Also	Recordset Object, CanMove Method, Move Method, MoveLast Method, MoveNext Method, MovePrevious Method.

MoveLast Method

MoveLast Method

Moves the pointer to the last string upon the result of the query.

Applies to objects: Recordset.

Syntax

[[**Let**] *RetVal* =] *object*.**MoveLast**()

The **MoveLast** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.

RetVal	Optional. A **Boolean** type variable, that indicates whether the pointer was moved to the necessary string.

Remarks

The ability of the pointer to move in such way is determined with the help of the CanMove method with the **cdbMoveLast** parameter.

Example

. . . .

See Also	Recordset Object, CanMove Method, Move Method, MoveFirst Method, MoveNext Method, MovePrevious Method.

MoveNext Method

MoveNext Method

Moves the pointer to the next line upon the result of the query.

Applies to objects: Recordset.

Syntax

[[**Let**] *RetVal* =] *object*.**MoveNext**()

The **MoveNext** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable, that indicates whether the pointer was moved to the necessary string.

Remarks

Such repositioning is always possible.

Example

. . . .

See Also	Recordset Object, Move Method, MoveFirst Method, MoveLast Method, MovePrevious Method.

MovePrevious Method

MovePrevious Method

Moves the pointer to the previous position upon the result of the query.

Applies to objects: Recordset.

Syntax

[[**Let**] *RetVal* =] *object*.**MovePrevious**()

The **MovePrevious** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable, that indicates whether the pointer was moved to the necessary string.

Remarks

The ability of the pointer to move in such way is determined with the help of the CanMove method with the **cdbMovePrevious** parameter.

Example

. . . .

See Also	Recordset Object, CanMove Method, Move Method, MoveFirst Method, MoveLast Method, MoveNext Method.

Move Method

Move Method

Moves the pointer in the desired direction upon the result of the query.

Applies to objects: Recordset.

Syntax

[[**Let**] *RetVal* =] *object*.**Move**([*Step*], [*Start*])

The **Move** method syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable, that indicates whether the pointer was moved to the necessary string.
Step	Optional. A **Long** type variable, indicates the number of strings to move by.
Start	Optional. A **Long** type variable, indicates the string number to move to.

Remarks

If *Start* is not defined or equals **0**, the pointer will be moved relative to the current string. The ability of the pointer to move in such way is determined with the help of the CanMove method with the **cdbMoveRelative** parameter.

If the *Start* and *Step* parameters are not defined or equal **0**, the pointer will be moved one string forward. The ability of the pointer to move in such way is determined with the help of the CanMove method with the **cdbMoveNext** parameter.

If the *Start* is defined and not equal to **0**, the pointer will be moved to the string described as *Start* + *Step*. The ability of the pointer to move in such way is determined with the help of the CanMove method with the **cdbMoveAbsolute** parameter.

The *Step* parameter can take positive or negative values.

Example

. . . .

See Also Recordset Object, CanMove Method, MoveFirst Method, MoveLast Method, MoveNext Method, MovePrevious Method.

OpenConnection Method

OpenConnection Method

Establishes connection with a database and creates a new Connection object.

Applies to objects: Workspace.

Syntax

[[**Set**] *RetVal* =] *object*.**OpenConnection**([*Name*], [*Exclusive*], [*ReadOnly*], [*ConnectionString*])

The **OpenConnection** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A Connection type variable.
Name	Optional. A **String** type variable. The name of the connection being created.
Exclusive	Optional. A **Boolean** type variable. A flag that indicates that database resources, used by the user will be blocked for other users. The default value is **FALSE**.
ReadOnly	Optional. A **Boolean** type variable. A flag, that indicates that the database is opened only for reading. The default value is **FALSE**.
ConnectionString	Optional. A **String** type variable. The connection initialization string.

Remarks

If the connection initialization string is omitted or doesn't contain the necessary parameters, the necessary information will be taken from the Workspace object.

Example

. . . .

See Also Connection Object, Workspace Object, OpenDatabase Method.

OpenDatabase Method

OpenDatabase Method

Establishes connection with a database and creates a new Database object.

Applies to objects: Workspace.

Syntax
[[**Set**] *RetVal* =] *object*.**OpenDatabase**([*Name*], [*Exclusive*], [*ReadOnly*], [*ConnectionString*])

The **OpenDatabase** method syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A Database type variable.
Name	Optional. A **String** type variable. The name of the connection being created.
Exclusive	Optional. A **Boolean** type variable. A flag that indicates that database resources, used by the user will be blocked for other users. The default value is **FALSE**.
ReadOnly	Optional. A **Boolean** type variable. A flag, that indicates that the database is opened only for reading. The default value is **FALSE**.
ConnectionString	Optional. A **String** type variable. The connection initialization string.

Remarks

If the connection initialization string is omitted or doesn't contain the necessary parameters, the necessary information will be taken from the Workspace object.

Example

. . . .

See Also Database Object, Workspace Object, OpenConnection Method.

OpenRecordset Method

OpenRecordset Method

Creates a new Recordset object that describes the results of an SQL query.

Applies to objects: Connection, Database, TableDef.

Syntax

[[**Set**] *RetVal* =] *object*.**OpenRecordset**([*Name*], [*CursorType*])

The **OpenRecordset** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A Recordset type variable.
Name	Optional. A **String** type variable. The name of the opened table, stored procedure, or an SQL query.
CursorType	Optional. A **Long** type variable. The pointer type. The default value is **cdbCursorForwardOnly**.

Remarks

The *Name* parameter may contain the name of the opened table, the stored procedure or an SQL query. If the method is applied to the TableDef object, the *Name* parameter is omitted and the method opens the table.

The *CursorType* variable defines the pointer type. Can have the following values:
cdbCursorForwardOnly - the pointer can move only forward and is not sensitive to changes in the database.
cdbCursorStatic - the pointer is not sensitive to changes in the database.
cdbCursorKeySet - the pointer is sensitive to updates in the database.
cdbCursorDynamic - the pointer is sensitive to all changes in the database.

For defining the supported pointer types, the CursorTypes mask is used.

Example

. . . .

See Also	Connection Object, Database Object, Recordset Object, TableDef Object, CursorTypes Property.

Parameters Method

Parameters Method

Provides access to the Parameters collection.

Applies to objects: QueryDef

Syntax

[[**Set**] *RetVal* =] *object*.**Parameters**()

The **Parameters** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A Parameters type variable.

Example

. . . .

See Also Parameters Object, QueryDef Object.

Properties Method

Properties Method

Provides access to the Properties collection.

Applies to objects: Connection, Database, DBEngine, Field, Index, QueryDef, Recordset, Relation, TableDef, Workspace.

Syntax

[[**Set**] *RetVal* =] *object*.**Properties**()

The **Properties** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A Properties type variable.

Example

. . . .

See Also	Connection Object, Database Object, DBEngine Object, Field Object, Index Object, Properties Object, QueryDef Object, Recordset Object, Relation Object, TableDef Object, Workspace Object.

QueryDefs Method

QueryDefs Method

Provides access to the QueryDefs collection of the Connection and Database objects.

Applies to objects: Connection, Database.

Syntax

[[**Set**] *RetVal* =] *object*.**QueryDefs**()

The **QueryDefs** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A QueryDefs type variable.

Example

. . . .

See Also	Connection Object, Database Object, QueryDefs Object.

Recordsets Method

Recordsets Method

Provides access to the Recordsets collection of the Connection and Database objects.

Applies to objects: Connection, Database.

Syntax

[[**Set**] *RetVal* =] *object*.**Recordsets**()

The **Recordsets** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A Recordsets type variable.

Example

. . . .

See Also Connection Object, Database Object, Recordsets Object.

Refresh Method

Refresh Method

Refreshes the contents of the collection.

Applies to objects: Fields, Indexes, Relations, TableDefs.

Syntax

object.**Refresh**()

The **Refresh** method syntax has these Elements:

Element	**Description**
object	Required. An instance of one of the object collections, listed above.

Remarks

The **Refresh** method is called automatically for all of the collections if necessary. The method synchronizes the contents of the collection with the corresponding database structure. Calling the **Refresh** method for the objects listed above guarantees that they will correspond to all latest changes.

Example

. . . .

See Also Fields Object, Indexes Object, Relations Object, TableDefs Object.

Relation Method

Relation Method

Provides access to the Relations collection of the Database object.

Applies to objects: Database.

Syntax

[[**Set**] *RetVal* =] *object*.**Relation**()

The **Relation** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A Relations type variable.

Example

. . . .

See Also Database Object, Relations Object.

RollbackTrans Method

RollbackTrans Method

Cancels a transaction.

Applies to objects: Connection, Database, Workspace.

Syntax

[[**Let**] *RetVal* =] *object*.**RollbackTrans**()

The **RollbackTrans** method syntax has these Elements:

Element	**Description**
object	Required. An instance of an object.
RetVal	Optional. **Boolean** variable, Shows whether we were able to cancel the transaction.

Remarks

All operations between BeginTrans and **RollbackTrans** will be cancelled.

After calling the **RollbackTrans** method the automatic transaction execution mode is restored. To form the next transaction, call BeginTrans again.

Depending on the driver and the database, the CommitTrans and **RollbackTrans** method may closes open pointers of Recordset objects.

Example

. . . .

See Also Connection Object, Database Object, Workspace Object, BeginTrans Method, CommitTrans Method.

RowsAffected Method

RowsAffected Method

The **RowsAffected** method returns the number of records, affected by the last operation.

Applies to objects: Recordset.

Syntax

[[**Let**] *RetVal* =] *object*.**RowsAffected**()

The **RowsAffected** method syntax has these Elements:

Element	Description
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Long** type variable, that gets the number of records.

Remarks

The **RowsAffected** method returns the number of records, affected by the AddNew, Delete and Update methods.

In some cases it's not possible to determine the number of records in the resulting multitude. The majority of data sources don't let determine the size of the resulting multitude when performing the **SELECT** operation, but determine the size successfully after **UPDATE**, **DELETE**, **INSERT**.

If the number of strings is unknown, the method returns **-1**.

Example

. . . .

See Also Recordset Object, GetRows Method.

TableDefs Method

TableDefs Method

Provides access to the TableDefs collection of the Database object.

Applies to objects: Database.

Syntax
[[**Set**] *RetVal* =] *object*.**TableDefs**()

The **TableDefs** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A TableDefs type variable.

Example
. . . .

See Also Database Object, TableDefs Object.

Update Method

Update Method

The **Update** method updates the current string in the table.

Applies to objects: Recordset.

Syntax
[[**Let**] *RetVal* =] *object*.**Update**()

The **MethodName** method syntax has these Elements:

Element	**Description**
object	Required. A reference to an instance of the object.
RetVal	Optional. A **Boolean** type variable that indicates whether the string could be updated.

Remarks

If the given object is the result of selection from the table fields, the method will take no action and return **FALSE**.

This method is called automatically when closing and repositioning the pointer.

Example

```
.......
```

See Also Recordset Object.

Workspaces Method

Workspaces Method

Provides access to the Workspaces collection of the DBEngine object.

Applies to objects: DBEngine.

Syntax

[[**Set**] *workspacesRet* =] *object*.**Workspaces**()

The **Workspaces** method syntax has these Elements:

Element	**Description**
object	Required. An instance of the DBEngine object.
workspacesRet	Optional. A Workspaces type variable.

Example

```
 ' Declaring and initializing variables
Dim engine As dbEngine
Set engine = new dbEngine
Dim wspace As Workspace
Set wspace = engine.CreateWorkspace("MyWorkspace")
'...
' Get the Workspace object collection
Dim wscoll As Workspaces
Set wscoll = engine.Workspaces()
'...
```

See Also DBEngine Object, Workspaces Object

Databases access Constants

Databases access Constants

Constants	Value	Descriptions
Databases types		
cddbBigInt	**16**	Long Numeric type(8 bytes)
cddbBinary	**9**	Binary type
cddbBoolean	**1**	Boolean type (2 bytes)
cddbByte	**2**	Byte type (1 byte).
cddbChar	**18**	Char type (1or 2 bytes)
cddbCurrency	**5**	Currency type (4 bytes)
cddbDate	**8**	Date type
cddbDecimal	**20**	Decimal type
cddbDouble	**7**	Double (8 bytes)
cddbGUID	**15**	Identifier
cddbInteger	**3**	Integer type (2 bytes)
cddbLong	**4**	Long integer type (4 bytes)
cddbMemo	**12**	Memo type (for large text or binary data)
cddbNumeric	**19**	Numeric type
cddbSingle	**6**	Float type (4 bytes)
cddbText	**10**	String type (no more than 256 characters)
cddbTime	**22**	Time type
cddbTimeStamp	**23**	Date and time
Parameters types		
cddbInput	**1**	Input
cddbInputOutput	**2**	Input/Output
cddbOutput	**4**	Output
Cursor types		
cddbCursorForwardOnly	**1**	The cursor can move only forward and is not sensitive to changes in the database.

cddbCursorKeyset	**2**	The cursor is sensitive to updates in the database.
cddbCursorDynamic	**4**	The cursor is sensitive to all changes in the database.
cddbCursorStatic	**16**	The cursor is not sensitive to changes in the database.
Cursor move direction		
cddbMoveNext	**1**	Allows using MoveNext
cddbMoveFirst	**2**	Allows using MoveFirst
cddbMoveLast	**4**	Allows using MoveLast
cddbMovePrevious	**8**	Allows using MovePrevious
cddbMoveAbsolute	**16**	Allows using Move for absolute positioning
cddbMoveRelative	**32**	Allows using Move for relative positioning
Transaction types		
cddbTransReadUncommited	**1**	Transaction doesn't isolate each other. There are following situations may occur: **Dirty Read**. Transaction 1 changes a row. Transaction 2 reads the changed row before transaction 1 commits the change. If transaction 1 rolls back the change, transaction 2 will have read a row that is considered to have never existed. **Nonrepeatable Read**. Transaction 1 reads a row. Transaction 2 updates or deletes that row and commits this change. If transaction 1 attempts to reread the row, it will receive different row values or discover that the row has been deleted. **Phantom**. Transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 generates one or more rows (through either inserts or updates) that match the search criteria. If transaction 1 reexecutes the statement that reads the rows, it receives a different set of rows.
cddbTransReadCommited	**2**	**"Dirty Read"** not allowed.
cddbTransRepeatableRead	**4**	**"Dirty Read"** and **"Nonrepeatable Read"** not allowed.
cddbTransSerializable	**8**	**"Dirty Read"**, **"Nonrepeatable Read"** and **"Phantom"** not allowed.

Conformance levels		
cddbLevel0	**0**	Minimal conformance level
cddbLevel1	**1**	Base conformance level
cddbLevel2	**2**	Extended conformance level
Driver type		
cddbODBC	**0**	ODBC driver

Trappable errors

Trappable errors

Trappable errors can occur while an execution is in progress. Some of these can also occur during compilation. At run time you can test and respond to trappable errors using the **On Error** statement and the **Err** function.

The following table lists trappable error messages and their detailed descriptions. Error number(#) is the value used to trap or return the error at run time.

#	Message	Description
1	*parser_message*	Occurs during compile time.
2	Syntax error	Occurs during compile time.
3	Return without GoSub	Occurs during run time.
4	Too many parameters in method or procedure '*procname*' call	Occurs during compile time.
5	Illegal method or procedure '*procname*' call	Occurs during compile time.
6	Overflow	
7	Out of memory	
8	Symbol '*symbol*' is not a constant. Constant is required	Occurs during compile time.
9	Subscript out of range or missing	
10	Duplicate definition '*symbol*'	Occurs during compile time.

11	Division by zero	
12	Expected function or variable	Occurs during compile time.
13	Type mismatch	
14	Out of string space	
15	Can't allocate symbol '*symbol*'	
16	Expression too complex	
17	Can't perform requested operation	
18	User interrupt occurred	
19	No Resume	
20	Resume without error	
21	Invalid event definition	
22	Invalid event param declaration	
23	Undefined label '*labelname*'	
24	Assignment isn't permitted	
25	Undefined symbol '*symbol*'	
26	Can't create object '*object*'	
27	Statement '*statement*' without 'Sub' ('Sub' is missing)	
28	Out of stack space	
29	Statement '*statement*' without 'For' ('For' is missing)	
30	Statement '*statement*' without 'Do' ('Do' is missing)	
31	Statement '*statement*' without 'While' ('While' is missing)	
32	Illegal statement '*statement*' after beginning of procedure ('End Sub' or 'End Function' is missing)	

33	Symbol '*symbol*' declaration without begriming of procedure ('Sub' or 'Function' is missing)	
34	Object '*object*' not an array	
35	Sub or Function not defined	
36	Statement '*statement*' without 'Function' ('Function' is missing)	
37	Undefined method or property '*name*'	
38	Property '*name*' is write-only	
39	Property '*name*' is read-only	
40	Property '*name*' is not found	
41	Can't open include '*filename*' or recycling include	
42	Invalid object type '*symbol*'	
43	Can't define Function '*procname*'	
44	Can't define Sub '*procname*'	
45	Invalid object reference	
46	Too many external library application clients	
47	Too many external library application clients	
48	Error in loading external library '*name*'	
49	Bad external library calling convention	
50	Specified external library procedure '*procname*' not found	
51	Internal error	
52	Bad file name or number	
53	File not found	

54	Bad file mode	
55	File already open	
57	Device I/O error	
58	File already exists	
59	Bad record length	
61	Disk full	
62	Input past end of file	
63	Bad record number	
64	Bad file name	
67	Too many files	
68	Device unavailable	
69	Procedure param '*param*' re-declaration (type mismatch)	
70	Permission denied	
71	Disk not ready	
74	Can't rename with different drive	
75	Path/File access error	
76	Path not found	
77	'Else' or 'End If' are missing	
78	'End If' is missing	
79	'Next' is missing	
80	'Loop' is missing	
81	'Wend' is missing	
82	'End Sub' is missing	
83	'End Function' is missing	
84	'End Select' is missing	
90	Database error: *message*	
91	Object variable not set	
92	For loop not initialized	
93	Invalid pattern string	

94	Invalid use of Null	

Glossary

Glossary

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

argument

A constant, variable, or expression passed to a procedure.

array

A set of sequentially indexed elements having the same intrinsic data type. Each element of an array has a unique identifying index number. Changes made to one element of an array don't affect the other elements.

Boolean data type

A data type with only two possible values, **True** (equal to 1 for arithmetical operations and -1 for logical operations) or **False** (0). **Boolean** variables are stored as 16-bit (2-byte) numbers.

by reference

A way of passing the address of an argument to a procedure instead of passing the value. This allows the procedure to access the actual variable. As a result, the variable's actual value can be changed by the procedure to which it is passed. Unless otherwise specified, arguments are passed by reference.

Byte data type

A data type used to hold positive integer numbers ranging from 0–255. **Byte** variables are stored as single, unsigned 8-bit (1-byte) numbers.

by value

A way of passing the value of an argument to a procedure instead of passing the address. This allows the procedure to access a copy of the variable. As a result, the variable's actual value can't be changed by the procedure to which it is passed.

comment

Text added to code that explains how the code works. In ConceptDraw Basic, a comment line can start with either an apostrophe (') or with the **Rem** keyword followed by a space.

comparison operator

A character or symbol indicating a relationship between two or more values or expressions. These operators include less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=), not equal (<>), and equal (=). Additional comparison operators include **Is** and **Like**. Note that **Is** and **Like** can't be used as comparison operators in a **Select Case** statement.

compile time

The period during which source code is translated to executable p-code.

compile-time error

An error that occurs when code is compiling.

constant

A named item that retains a constant value throughout the execution of a program. A constant can be a string or numeric literal, another constant, or any combination that includes arithmetic or logical operators. There are many embedded constants. Additional constants can be defined by the user with the **Const** statement. You can use constants anywhere in your code in place of actual values.

data type

The characteristic of a variable that determines what kind of data it can hold. Data types include **Byte**, **Boolean**, **Integer**, **Long**, **Single**, **Double**, **Date**, **String**, **Object**, **Variant** (default), and specific types of embedded objects.

Date data type

A data type used to store dates and times as a real number. Date variables are stored as 64-bit (8-byte) numbers. The value to the left of the decimal represents a date, and the value to the right of the decimal represents a time.

date expression

Any expression that can be interpreted as a date, including date literals, numbers that look like dates, strings that look like dates, and dates returned from functions. A date expression is limited to numbers or strings, in any combination, that can represent a date from January 1, 100 – December 31, 9999.

Dates are stored as Element of a real number. Values to the left of the decimal represent the date; values to the right of the decimal represent the time. Negative numbers represent dates prior to December 30, 1899.

declaration

Nonexecutable code that names a constant, variable, or procedure, and specifies its characteristics, such as data type. For external procedures (DLL procedures), declarations specify names, libraries, and arguments.

Double data type

A data type that holds double-precision floating-point numbers as 64-bit numbers in the range -1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values.

dynamic-link library (DLL)

A library of routines loaded and linked into applications at run time. DLLs are created with other programming languages such as C, MASM, or FORTRAN. This term is mostly used on the Windows platform, on the Mac platform "Shared Library"is used instead.

Empty

A state of a variable. Indicates that no beginning value has been assigned to a variable. An **Empty** variable is represented as 0 in a numeric context, a zero-length string ("") in a fixed-length string context or as **Null** in a variable-length string context and in a object context.

error number

A whole number in the range 0 – 65,535 that corresponds to the error number returned by **Err**() function.

execution level

The level of definition and execution of a script, that corresponds to the object, owning the script. ConceptDraw supports the following execution levels: Application level, Document level, Page level, Shape level. Any execution level contains at least a built-in module with code in ConceptDraw Basic.

expression

A combination of keywords, operators, variables, and constants that yields a string, number, or object. An expression can be used to perform a calculation, manipulate characters, or test data.

file number

Number used in the **Open** statement to open a file.

Integer data type

A data type that holds integer variables stored as 2-byte whole

numbers in the range -32,768 to 32,767.

InternalUnit

Internal units of measure in ConceptDraw, 1 InternalUnit = 0.1 mm.

keyword

A word or symbol recognized as Element of the ConceptDraw Basic programming

language; for example, a statement, function name, or operator.

line label

Used to identify a single line of code. A line label can be any combination of characters that starts with a letter and ends with a colon (:). Line labels are not case sensitive and must begin in the first column.

line number

Used to identify a single line of code. A line number can be any combination of digits that is unique within the module where it is used. Line numbers must begin in the first column.

locale

The set of information that corresponds to a given language and country. The code locale setting affects the language of terms such as keywords and defines locale-specific settings such as the decimal and list separators, date formats, and character sorting order.
The system locale setting affects the way locale-aware functionality behaves, for example, when you display numbers or convert strings to dates.

Long data type

A 4-byte integer ranging in value from -2,147,483,648 to 2,147,483,647. The **Long** data type is also used to represent enumerated values.

module

A set of declarations and definitions of procedures, variables and also a set of ConceptDraw Basic instructions, united by the common global area of visibility of procedures and global variables.

module level

Describes code in the Declarations section of a module. Any code outside a procedure is referred to as module-level code.
Module level defines global visibility area of variables and procedures.

Null

A value, equal to zero. It's used to assign zero links to objects and strings of variable length. All non-initialized variables take the **Null** value together with the **Empty** state. In ConceptDraw Basic the **Null** value is also equivalent to the **Nothing** value.

numeric expression

Any expression that can be evaluated as a number. Elements of an expression can include any combination of keywords, variables, constants, and operators that result in a number.

numeric type

Any intrinsic numeric data type (**Byte**, **Boolean**, **Integer**, **Long**, **Single**, **Double**, or **Date**) or any **Variant** numeric subtype (**Integer**, **Long**, **Single**, **Double**, **Date**, **Boolean**, or **Byte**).

Object data type

A data type that represents any **Object** reference. **Object** variables are stored as 32-bit (4-byte) addresses that refer to objects.

object type

A type of embedded object exposed by an application, for example, **Application**, **Document**, **Page** and **Shape**.

object variable

A variable that contains a reference to an object.

parameter

Variable name by which an argument passed to a procedure is known within the procedure. This variable receives the argument passed into the procedure. Its scope ends when the procedure ends.

pixel

The smallest element that can be displayed on a screen or printer. Pixels are screen-dependent. Contrast twip.

point

A unit of measurement for type whereby 12 points equal 1 pica, and 6 picas equal 1 inch; thus, 1 point equals 1/72 inch. See also twip.

print zone

Print zones begin every 14 columns. The width of each column is an average of the width of all characters in the point size for the selected font.

procedure

A named sequence of statements executed as a unit. For example, **Function** and **Sub** are types of procedures.

procedure level

Describes statements located within a **Function** or **Sub** procedure. Declarations are usually listed first, followed by assignments and other executable code.
Note that module-level code resides outside a procedure block.
Procedure level defines the local visibility area of variables.

property

A named attribute of an object. Properties define object characteristics such as size, color, and location, or the state of an object, such as enabled or disabled.

run time

The time during which code is running. During run time, you can't

edit the code.

run-time error

An error that occurs when code is running. A run-time error results when a statement attempts an invalid operation.

seed

An initial value used to generate pseudo-random numbers. For example, the **Randomize** statement creates a seed number used by the **Rnd** function to create unique pseudo-random number sequences.

Single data type

A data type that stores single-precision floating-point variables as 32-bit (4-byte) floating-point numbers, ranging in value from -3.402823E38 to -1.401298E-45 for negative values, and 1.401298E-45 to 3.402823E38 for positive values.

sort order

A sequencing principle used to order data, for example, alphabetic, numeric, ascending, descending, and so on.

statement

A syntactically complete unit that expresses one kind of action, declaration, or definition. A statement generally occupies a single line, although you can use a colon (:) to include more than one statement on a line.

String data type

A data type consisting of a sequence of contiguous characters that represent the characters themselves rather than their numeric values. A **String** can include letters, numbers, spaces, and punctuation. The String data type can store fixed-length strings ranging in length from 0 to approximately 63K characters and dynamic strings ranging in length from 0 to approximately 2 billion characters. In any case **String** stores text in the Unicode encoding, each symbol taking 2 bytes.

string expression

Any expression that evaluates to a sequence of contiguous characters. Elements of a string expression can include a function that returns a string, a string literal, a string constant, a string variable, a string **Variant**, or a function that returns a string **Variant** (**VarType** 8).

twip

A unit of measurement, implemented as 1/20 of a point, or 1/1440 of an inch. There are 567 twips to a centimeter. Twips are screen-independent measurements. See also point. Contrast pixel.

unit

A unit of measure in ConceptDraw. Units are screen-independent measurements. 1 millimeter contains 10 units.

variable

A named storage location that can contain data that can be modified during program execution. Each variable has a name that uniquely identifies it within its scope. A data type can be specified or not.

Variable names must begin with an alphabetic character, must be unique within the same scope, can't be longer than 32 characters, and can't contain an embedded period or type-declaration character.

Variant data type

A special data type that can contain numeric, string, or date data as well as the special values **Empty** and **Null**. The **Variant** data type has a numeric storage size of 16 bytes and can contain data up to the range of a Double, or a character storage size of 22 bytes (plus string length), and can store any character text. The **VarType** function defines how the data in a **Variant** is treated. All variables become **Variant** data types if not explicitly declared as some other data type.

CS Odessa
US/Canada/Mexico
Technical Support:
+1 (877) 441 - 1150 ext 4 Toll Free
+1 (408) 441 - 1150 ext 4

Sales:
+1 (877) 441 - 1150 ext 3 Toll Free
+1 (408) 441 - 1150 ext 3

Rest of the World
Technical Support/Sales:
+44 (203) 514 - 70 - 40

support@conceptdraw.com
sales@conceptdraw.com