A Look at The Features of 4D View Pro

By Tai Bui, Technical Services Engineer, 4D Inc.

Technical Note 19-02

Table of Contents

Introduction

4D databases can contain a large amount of data. To utilize the data, it can be useful to have control over how the data is displayed to a user. One of the features that 4D provided, to display data with control over it, is 4D View. Moving forward, to meet the changing standards of hardware and software, a new set of features called 4D View Pro was created with the goal of fulfilling the typical uses of the legacy 4D View and attempting to improve on them.

This Technical Note will explore the features of 4D View Pro as of v17R4. 4D View has multiple targeted uses, and as such 4D View Pro provides several features to attempt to allow implementations of these uses and more. Examples and explanations on how to use 4D View Pro and implement some features will be provided along with a sample database with some demos of the features.

Overview

4D View is deprecated and is planned to be succeeded by 4D View Pro in the same fashion as 4D Write and 4D Write Pro for the same reasons. The reasoning is due to the architecture of the technology used to design the legacy versions of these features becoming outdated and it would be better to rewrite the features using current technology.

4D View

4D View is an optional plugin that has been provided with the installation of 4D. The typical use of the plugin is used in 4D for multiple ways to display data. The main typical uses of 4D View are to display data in a spreadsheet, display data in a list, create a report, and to display a calendar. However, 4D View was designed using a 32-bit architecture. With the shift to 64-bit being the standard and rumors of 32-bit support being dropped, 4D View would not be able to be run anymore without changes.

4D View Pro

4D View Pro is a native feature of 64-bit 4D however, it is contained in a component. While it shares a similar name to 4D View, it is not 4D View. Instead of going through 4D View and updating and changing everything to use 64-Bit equivalents, it was more efficient to rewrite a new feature from the bottom up based on 64-bit technology. During this rewrite, it was important to make sure that it is a proper successor to 4D View and that all aspects of 4D View would be able to be handled by 4D in some way. With the differences in architecture 4D View Pro embraces the new Object data type and uses it as the basis for maintaining the data of a 4D View Pro document. 4D View Pro also provides its own set of commands which are different from 4D View commands.

In short, the legacy 4D View was designed for a 32-Bit environment and with the shift to a 64-bit environment being standard and the deprecation of 32-Bit technology, 4D View Pro is the replacement that will be compatible with a 64-Bit environment. This is important is the database is planned to be updated to 4Dv17 R5 and v18 and above since 32-Bit versions will no longer be provided and maintained for compatibility purposes.

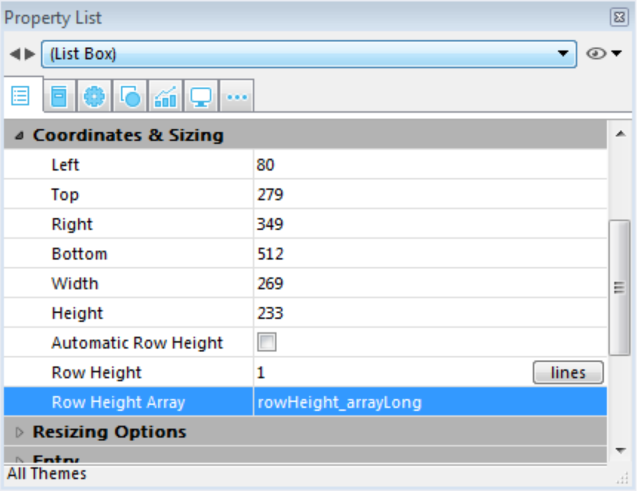
A sample database is provided with this Technical Note. Although the sample database will come with short explanations of each demoed feature and commented code, it is beneficial to first read through the Technical Note first to understand the features of 4D View Pro as the explanations are brief and under the assumption that this document has been reviewed first. The database uses features introduced in v17R4 and will need a 4D View Pro license to actually run the features.

4D View Pro List Box Features

List boxes are a very powerful feature of 4D and have many features and customization possibilities. A list box is used to display data to the user for several uses. These uses of list boxes share some overlap with the intended use of 4D View to display data, as such some features have been added to list boxes so that they can be used as an alternative way to display data in the absence of the legacy 4D View. There are three main features that have been added to list boxes that also require the addition of a 4D View/4D View Pro license to use.

Variable Row Height

A feature that 4D View Pro adds to a list box is the ability to have independent varying row heights. This feature is only available to array-based list boxes. Without this feature, the height of all the rows are the same and based on the Row Height property. When this feature is utilized a Longint Array is assigned to the Row Height Array property. The array will automatically have the same number of elements as there are rows and by default, all elements will be set to 0. When the element is set to 0 the row will use the default height assigned to the Row Height property, but when an actual value is set it will adjust the associated row’s height to the value in pixels. This new feature allows manual control over each individual row’s height allowing the display of data to be improved, especially when the displayed data has a large variance of sizes in terms of display, such as images or large text.

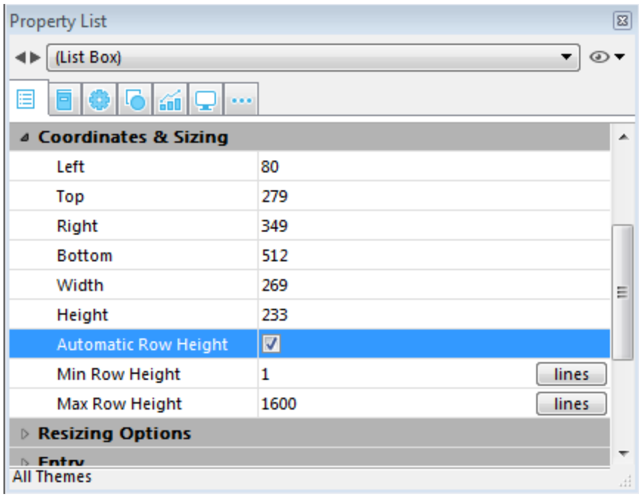


Left	80
Top	279
Right	349
Bottom	512
Width	269
Height	233
Automatic Row Height	☐
Row Height	1
Row Height Array	rowHeight_arrayLong

Image 1: Row Height Array Property of a List Box

Automatic Row Height

The next feature that 4D View Pro grants to a list box is an expansion to the idea of the variable row height which is the automatic row height. This feature, list the variable row height feature, is only available to array-based list boxes. The feature automates the process by allowing 4D to automatically adjust the rows to accommodate all the data in all columns. The means that each row will try to adjust their heights to display all the data of the cell with the largest requirement per row. To prevent issues with the row being too large or too small, when this property is enabled there are two associated properties that define for the minimum height and maximum height of a row replacing the definition for the default row height and the row height array properties as shown below.

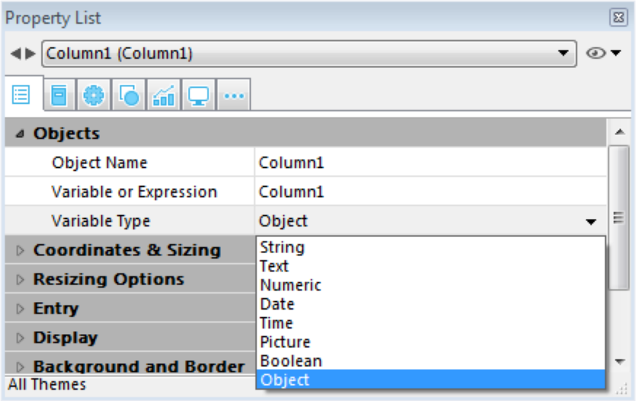


Left	80
Top	279
Right	349
Bottom	512
Width	269
Height	233
Automatic Row Height	☑
Min Row Height	1
Max Row Height	1600

Image 2: Automatic Row Height Property of a List Box

Object Type Arrays

The last feature that 4D View Pro provides to a list box is allowing the object data type array to be associated with a list box column. Object Type Arrays are powerful in the sense that they allow the display of multiple data types. This freedom allows list boxes to display a larger number of things and are not limited to listing rows of items consistent with the data used for a column. With an object array assigned to a column, the column can display a numerical value, a Date, a Boolean, and a Text without any errors. With some effort, a list box can be more than a list box displaying a grid of varying information.

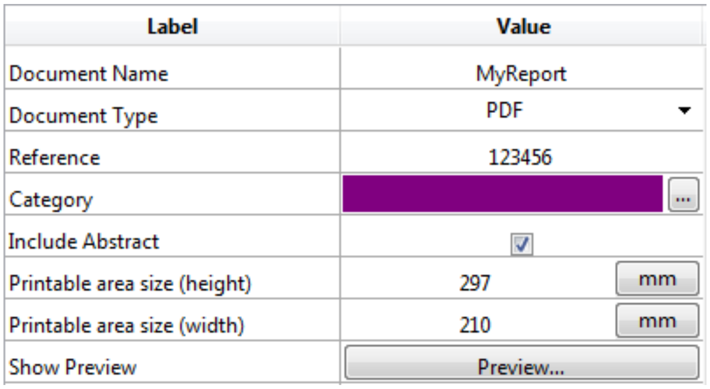


Objects

Object Name	Column1
Variable or Expression	Column1
Variable Type	Object

Image 3: Variable Type Property of a List Box Column Set to Object

Another feature of applying an object type array to a list box is the ability to access specialized Widgets to customize each cell. By specifying and defining specific properties in an object of the array additional features can be added to each cell besides displaying data. As shown in the list box below, dropdowns, buttons, and more can be added to each cell.



Label	Value
Document Name	MyReport
Document Type	PDF ▾
Reference	123456
Category	🟪 ...
Include Abstract	☑
Printable area size (height)	297 mm
Printable area size (width)	210 mm
Show Preview	Preview...

Image 4: Example of an Object Array Based List Box with Widgets

The structure of each object in the array can vary but there is a list of relevant properties that will format the cells. The only mandatory attribute is "valueType", this property allows the object to define what type and features the cell presents. The applicable string values for this property are:

- "text" – basic text input, drop-down menu, or combo box
- "real" – numeric input with separators, drop-down list, or combo box
- "integer" – basic numeric input, text drop-down list, or combo box
- "Boolean" – checkbox or Boolean drop-down list
- "color" – background color or text representation of color
- "event" – a labeled button

The other properties of the objects provide additional info to fine tune each cell if needed. These properties are and the description of their values:

- "value" – the value of the cell
- "min" – the minimum value of a numeric value type
- "max" – the maximum value of a numeric value type
- "behavior" – an integer value type to display a three-state checkbox
- "requiredList" – array to be used as a drop-down list
- "requiredListReference" – 4D list reference for drop-down list
- "requiredListName" – 4D list name for drop-down list
- "choiceList" – array to be used as a combo box
- "choiceListReference" – 4D list reference for a combo box
- "choiceListName" – 4D list reference for a combo box
- "saveAs" – when using a list, pass "value" to save the value of the selected item or "reference" to save the item number of the selected item
- "unitList" – array for the list of units of measurement for the value
- "unitsListReference" – 4D list reference for units of measurement
- "unitsListName" – 4D list name for units of measurement
- "unitReference" – The item number of the selected unit of measurement
- "alternateButton" – Provides an appended button labeled with an ellipsis which triggers an On Alternate Click event to allow specific code to be executed.

4D View Pro provides a couple of features to list boxes that allow more freedom to the developer when deciding how to display data. With list boxes already utilized to display lists of data and containing many configuration properties, three more specific to 4D View Pro exists. The first two deal with allowing individual rows to have varying heights. This allows data to be displayed in a neater fashion without having them all uniform causing cutoffs or displaying large white space. The Row Height Array property allows manual fine control over the height but will require more effort to implement, while the similarly functioning Automatic Row Height property is easier to enable but provides less control with the only control being the minimum and maximum heights. The third feature is enabling the use of object type arrays for a column. This allows the use of the list box's grid-like structure to be utilized in new ways that could not be done before.

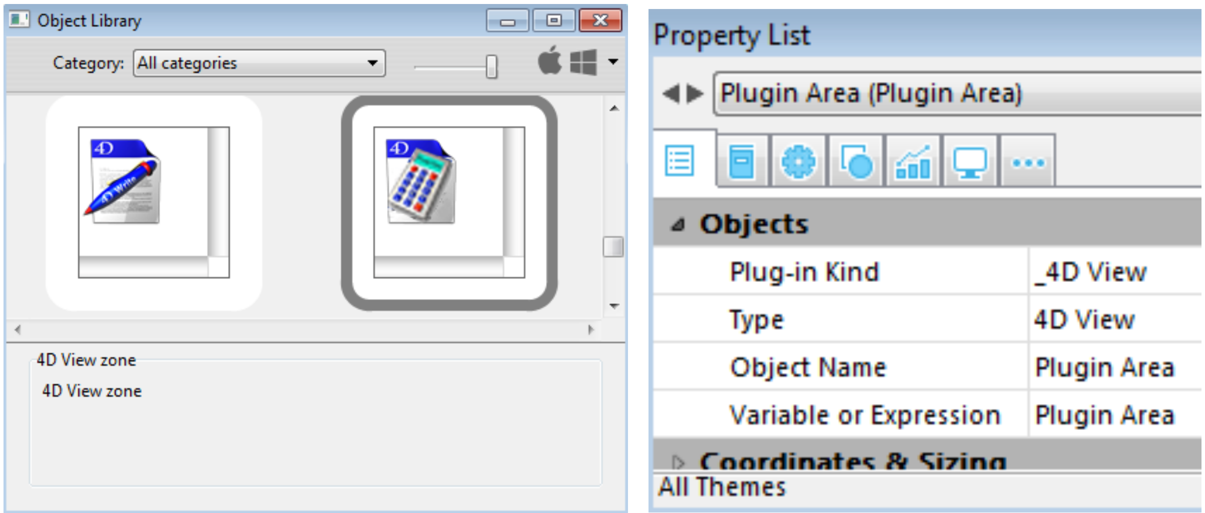
4D View Pro Area

4D View Pro provides a new updated form object which is the 4D View Pro Area. Between 4D View Pro and the legacy 4D View areas, there are some differences between them. The 4D View Pro area comes with several features already implemented into the toolbar for ease of use however, the functionally is targeted towards manipulating the contents of the 4D View Pro document. However, there are several 4D View Pro commands that can be used to implement features for manipulating the document itself.

Implementing the 4D View Pro Area

To start with the implementations of the areas are different. With the legacy 4D View, the area can be accessed in one of two ways. The first is the Tools menu which has a 4D View item to open a window dedicated to the 4D View area. This menu is available in design mode. The second way to use a 4D View area, which is also the same way to use a 4D View Pro area is to add it to a Form and run the form. Since 4D View is a plugin, the 4D View area is added by adding a Plugin Area and selecting the 4D View type.

4D View Pro is an integrated feature that is contained in a component. This allows 4D View Pro to have its own object called a 4D View Pro area and it just simply needs to be added to a form. If the component is not available, the button will still be displayed, but it will not be able to add a working 4D View Pro area. The dedicated object also has its own unique properties that the legacy 4D View was not able to have. These two properties are the User Interface, which is to enable or disable the toolbar by default, and Show Formula Bar, which is to enable or disable the formula bar by default.



Category: All categories

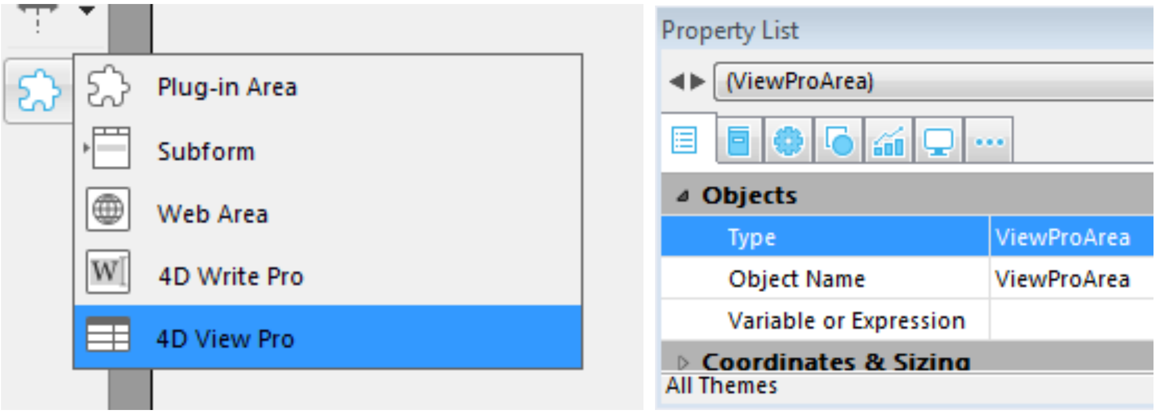
4D View zone

4D View zone

Plugin Area (Plugin Area)

Plug-in Kind	_4D View
Type	4D View
Object Name	Plugin Area
Variable or Expression	Plugin Area

Image 5: Accessing Legacy 4D View Area and its Property List



Type	ViewProArea
Object Name	ViewProArea
Variable or Expression	

Image 6: Accessing 4D View Pro Area in Form Editor and its Property List

4D View Pro Toolbar

Running the two areas of 4D View and 4D View Pro there have been changes visually. The legacy 4D View area uses a classic look that would match more with the classic OSes while the 4D View Pro area has an updated look using a flat design which has been a frequent trend of UI design. Another update that can be quickly noticed is the change of the toolbars which have been reduced of clutter by using a ribbon design to group similar functions into separate tabs. As mentioned the functionality of the toolbar is targeted at manipulating the contents of the 4D View Pro document.

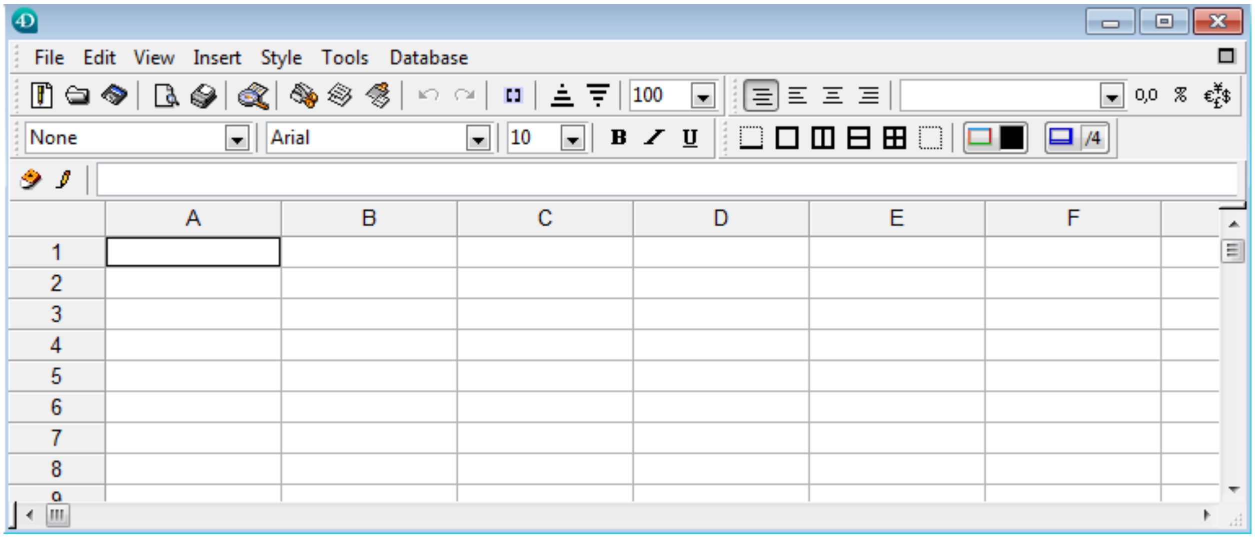


Image 7: The Running Legacy 4D View Area

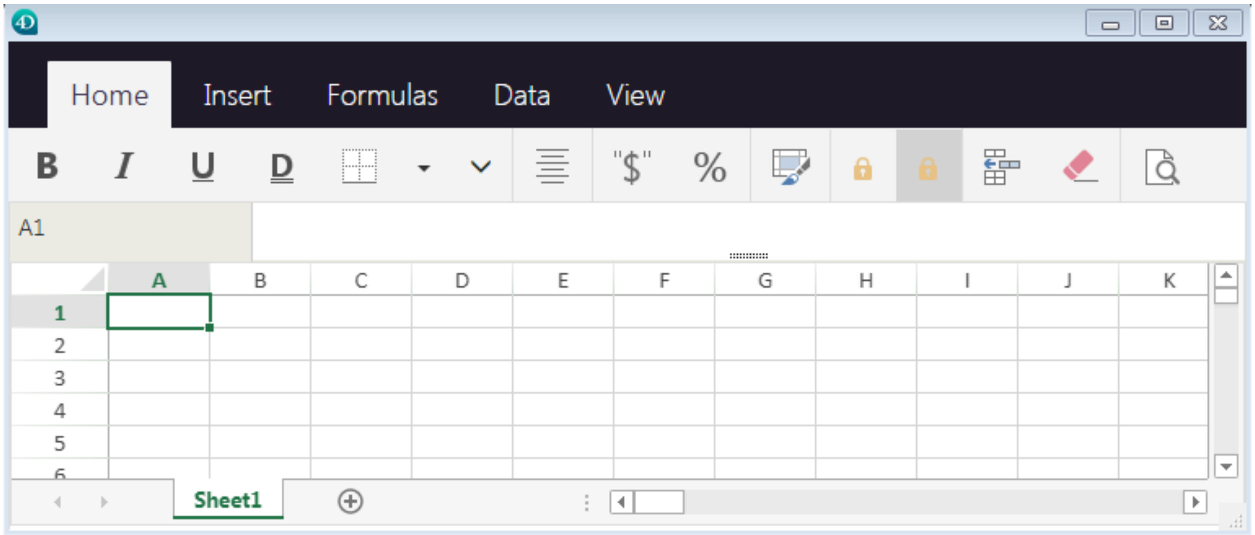


Image 8: The Running 4D View Pro Area

Manipulation of the document itself is not readily available as a prebuilt toolbar however, it can be implemented using the new 4D View Pro family of commands. The 4D View Pro commands all start with VP. There are commands to create a new document, open an existing document, and saving the current document. With these commands, custom features can be implemented to provide users with a way to handle the 4D View Pro document.

Using the 4D View Pro Area

Using the 4D View Pro Area has not changed much compared to the classic 4D View. Users with any familiarity with spreadsheet applications should find the feature easy to pick up and intuitive as with how 4D View was. It is also important to note that while 4D View Pro provides similar features and functionality to a spreadsheet application, it is not considered a full-featured spreadsheet application and as such may not have some of the unique features of that a standalone spreadsheet application might have.

There are some new additions to 4D View Pro that the legacy 4D View did not have. One new feature is a default contextual menu for the cells of the 4D View Pro area has been implemented. The contextual menu provides expected text manipulation functions and some functions specific to the 4D View Pro area.

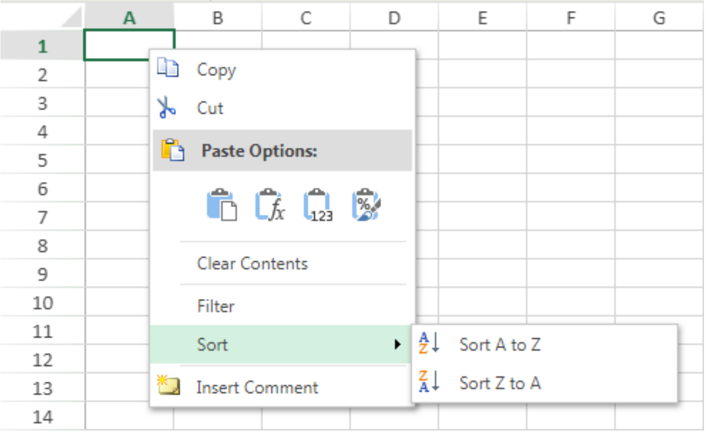


Image 9: The Contextual Menu of the 4D View Pro Area

The other new feature is the ability to have multiple sheets in a 4D View Pro document and the ability to navigate between them using the tab controls at the bottom like some other spreadsheet applications. The 4D View Pro area retains the functionality of the legacy 4D View while providing a new updated look with some additional features.

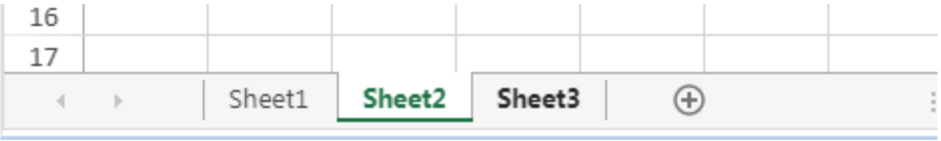


Image 10: Sheet Selection Function at the Bottom of 4D View Pro Area

Manipulating the 4D View Pro Document

4D View Pro has its own family of commands. Because 4D View Pro is not 4D View it cannot use the 4D View family of commands provided by the 4D View plugin. While the introduction of 4D View Pro came with a couple of commands, 4Dv17 R4 introduces many more commands for 4D View Pro. 4D View Pro is a component and as such the list of commands are listed under the component methods of 4D View Pro. The first introduced set of commands provided mainly control over the 4D View Pro document that should provide all the features available to the legacy 4D View and expected functions.

Converting a 4D View Document to a 4D View Pro Document

A nice feature provided with 4D View Pro is the VP Convert from 4D View command which will automatically convert a 4D View document in the form of a Blob to a 4D View Pro document in the form of an object. The command will attempt its best at generating a 1 to 1 conversion, but there is a possibility that some things may not be properly converted.

Some of the items not converted exactly 1 to 1:

- **Double-bar** – The Double-bar border in 4D View allowed multiple styles of the bars, which 4D View Pro only has one style and thus after a conversion, all double-bars will not have their original style but will be converted with the new style.
- **Splitter** – Splitters are not converted over.
- **Some Styles** – Deprecated styles are not converted over and Conditional Styles are not supported.
- **Rotation Styles** – Rotation styles of text are not converted over
- **Some Formats** – Cell formats that may rely on external system settings may not convert properly.
- **User Defined Formats** – User-defined formats are not supported.
- **Controls** – Controls created with PV SET CELL CONTROL are not supported
- **Multiple format Pictures** – Pictures with multiple formats will be converted with the most appropriate codec.
- **Pictures with Truncated Centered / Replicated Format** – Pictures with truncated centered / replicated format are currently not converted
- **Dynamic Links** – Cells or columns linked to fields or variables are not supported
- **Some Project Methods** – Project methods with a non-compliant method name will be converted in the form of UNSUPPORTED_4DMETHOD_NAME("<method name>",param1,...paramN).
- **Some Commands** – 4D commands that are not part of the authorized list are converted to: UNSUPPORTED_4DCOMMAND(<command name>,param1,...,paramN).
- **Variables** – 4D project methods must be used to access variable values. Variables in formulas are converted to UNSUPPORTED_VARIABLE("<variable name>").
- **Some Fields** – If a field or table name is not ECMA compliant, converted to UNSUPPORTED_TABLE_FIELD_TITLE("virtual structure name").

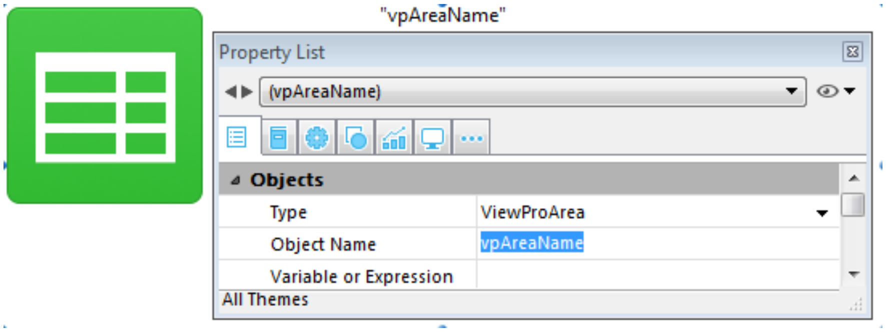
The feature is simple to use, below is an example of using the feature in a few lines of code:

```
C_BLOB($legacyView_blob)
C_OBJECT($viewProDoc_OB)
DOCUMENT TO BLOB(“myViewDocument.4PV”;$legacyView_blob)
$viewProDoc_OB:= VP Convert from 4D View
```

The code generates a 4D View Pro document in the object, which can then be saved somewhere or displayed in a 4D View Pro Area. The command does not overwrite the old document and as such, it is possible to maintain both copies just in case there are differences.

Opening a Document

4D View Pro can open a new document or an existing document in a couple of ways with native commands. To open a new document in a 4D View Pro area the command ***VP NEW DOCUMENT*** can be used passing the form area object's name as the first and only parameter. This will display a new default and empty document replacing anything previously displayed in the area.



Type: ViewProArea
Object Name: vpAreaName
Variable or Expression:

Image 11: For examples, 4D View Pro Form Area name is "vpAreaName"

The following code will display a new default and empty document replacing anything previously displayed in the area.

```
VP NEW DOCUMENT (“vpAreaName”)
```

There are two ways to open an existing document based on the two form factors that a 4D View Pro document may be stored as. One form is when the document is stored in an object. To load the document in a 4D View Pro area, the ***VP IMPORT FROM OBJECT*** command can be used, passing in the form area object's name as the first parameter and the object variable containing the document as the second parameter.

```
 //[Table_1]vpDocs is an Object type field
 // containing View Pro Documents

VP IMPORT FROM OBJECT (“vpAreaName”;[Table_1]vpDocs)
```

The other form of a 4D Document is an on-disk document format. To open documents on-disk, the command ***VP IMPORT DOCUMENT*** is used with a similar format to the prior command except with the document path as the first parameter. The command even works with .xlsx files.

```
VP IMPORT DOCUMENT (“vpAreaName”;”myVPdoc.4VP”)
```

These are the three ways to open a 4D View Pro document in a 4D View Pro area. While there are not readily available buttons in the View Pro toolbar, these commands can be used to implement opening functions when needed.

Saving a Document

Saving a 4D View Pro document is very similar to opening an existing 4D View Pro document with two separate commands for the two forms of storage. To store the document in an object, the ***VP Export to object*** command can be used with takes in the View Pro form area object's name as the only parameter and output the document to an object.

```
C_OBJECT($vpDoc_OB)
$vpDoc_OB:=VP Export to object("vpAreaName")
```

The ***VP EXPORT DOCUMENT*** command will allow a 4D View Pro document to be stored onto an external file on disk by passing in the 4D View Pro area containing the document as the first parameter and the file path name as the second parameter. The command is even able to export to .xlsx formatted files.

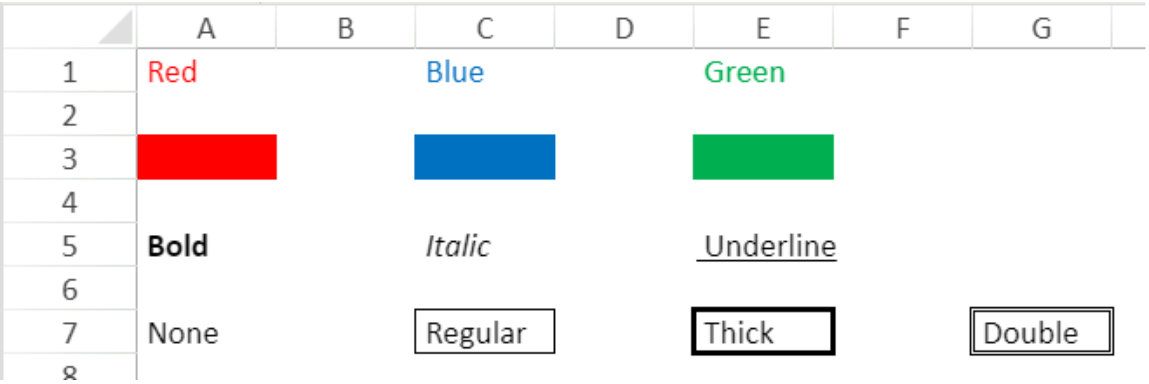
```
VP EXPORT DOCUMENT (“vpAreaName”;”myVPdoc.4VP”)
```

These commands will allow the ability to create, open, and save 4D View Pro documents. While they are not available as standard functions in the 4D View Pro area, a custom UI can be implemented to provide these features if needed. With the ability to save the documents in the database in objects these features may or may not be required.

4D View Pro Document Structure

A 4D View Pro document's structure is defined and stored in an object data type. This feature allows the document to be easily passed around in the 4D application due to the increasingly useful object data type which allows the document to be stored in table fields and in object type variables. If desired, the schema of a 4D View Pro document can be used to create or manipulate the 4D View Pro document through the object data.

As mentioned, 4D View Pro documents can have several sheets. Sheets are like pages of the 4D View Document with each being an individual spreadsheet. Like a spreadsheet, each sheet is comprised of a grid of rows and columns forming cells. Each cell contains data with the possibility of the strings being multi-styled. In addition to the contents, each cell's gridlines can also contain multiple styles.



	A	B	C	D	E	F	G
1	Red		Blue		Green		
2							
3	🟥		🟦		🟩		
4							
5	**Bold**		*Italic*		Underline		
6							
7	None		Regular		Thick		Double

Image 12: Example of some styling for individual cells.

4D View Pro Ranges

A 4D View Pro document is manipulated programmatically by specifying and targeting specific portions of the document, which can be called ranges. In 4D View Pro, ranges are a group of cells. By specifying which cell or cells in a range actions can then be performed on them. The columns are labeled in numerical order starting from 1 and the rows are labeled using the alphabet. When targeting the columns and rows, they are both identified by indexes starting from 0. Ranges can be created with some of the commands introduced in 4Dv17 R4.

Range Objects Structure

Ranges are maintained as object data types. The range objects are comprised of two properties. The first property is "area" which contains the name of the 4D View Pro area. The second property is a collection of ranges for the 4D View Pro area.

Each range collection can contain the following properties:

- name – the identifying name of the range
- sheet – the sheet number in the document
- row – the row index of the sheet
- rowCount – the number of rows from the index
- column – the column index of the sheet
- columnCount – the number of columns from the index

Creating Range Objects

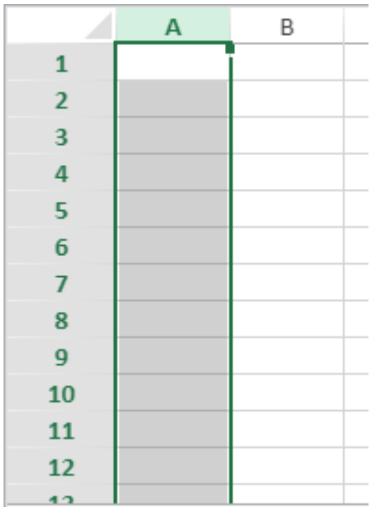
Generating ranges is simplified with the new commands. Like most of the 4D View Pro commands, the following commands all use the target View Pro area as the first parameter. The commands also have a last optional parameter to specify a certain sheet which defaults to the currently active sheet. Sheets are also identified by indexes starting from 0.

- ***VP All*** – Creates a range of all cells.

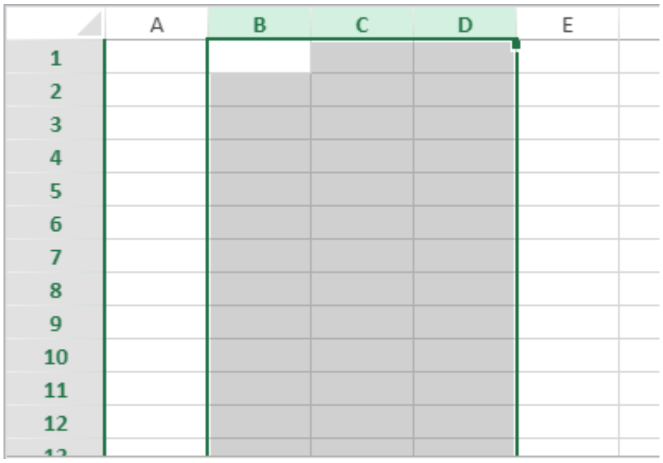
```
// All cells in current sheet
$vpRange1_OB:=VP All("vpAreaName")

// All cells in the first[0] sheet
$vpRange2_OB:=VP All("vpAreaName";0)
```

- ***VP Column*** – Creates a range of columns starting at the index specified in the second parameter with an optional parameter to specify the number of columns from the index.

```
// All cells first[0] column
$vpRange1_OB:=VP Column("vpAreaName";0)
```

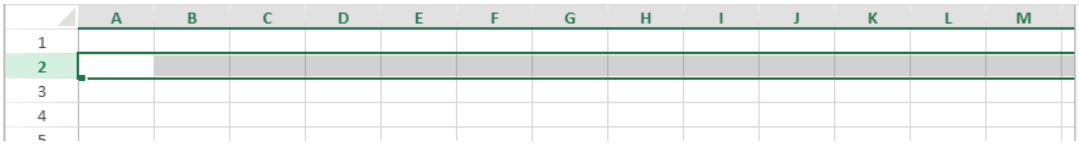


```
// All cell in 3 columns starting from second[1] column
$vpRange2_OB:=VP Column("vpAreaName";1;3)
```

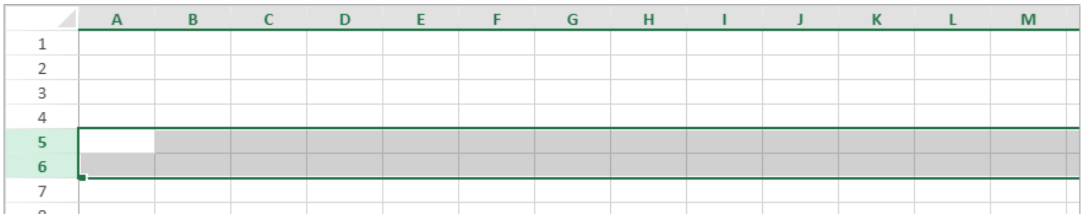


- ***VP Row*** - Creates a range of rows starting at the index specified in the second parameter with an optional parameter to specify the number of rows from the index.

```
// All cells second[1] row
$vpRange1_OB:=VP Row(“vpAreaName”;1)
```

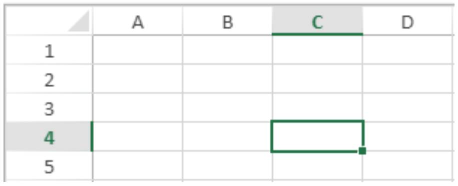


```
// All cells 2 in rows starting from fifth[4] row
$vpRange1_OB:=VP Row(“vpAreaName”;4;2)
```



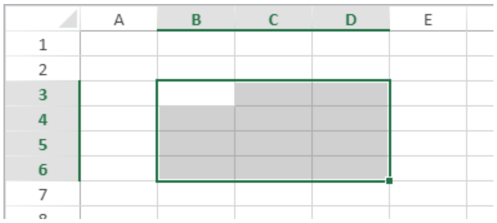
- ***VP Cell*** – Creates a range with the cell specified by the column index passed in the second parameter and the row index passed in the third parameter.

```
// Selects the cell at third[2] column and fourth[3] row
$vpRange_OB:=VP Cell(“vpAreaName”;2;3)
```



- ***VP Cells*** – Creates a range of starting with the cell specified by the column index passed in the second parameter and the row index passed in the third parameter up to the number of columns passed in the fourth parameter and the number of rows passed in the fifth parameter.

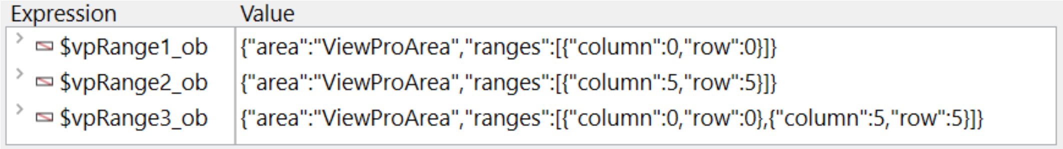
```
// Selects cells starting at second[1] column and third[2]
// row ending 3 columns and 4 rows out
$vpRange_OB:=VP Cells(“vpAreaName”;1;2;3;4)
```



Manipulating Range Objects

After creating the 4D View Pro ranges they may need to be manipulated to properly track and handle them. Each time a range is created, it is contained in its own range object. It is possible to combine range objects so that a range object can contain multiple sets of ranges in the range collections. This can be done with the ***VP Combine ranges*** command which requires at least two parameters for each range object to combine. The result will be one range object with a collection of the combined ranges.

```
$vpRange1_ob:=VP Cell ("ViewProArea";0;0)
$vpRange2_ob:=VP Cell ("ViewProArea";5;5)
$vpRange3_ob:=VP Combine ranges($vpRange1_ob;$vpRange2_ob)
```



Expression	Value
$vpRange1_ob	{"area":"ViewProArea","ranges":[{"column":0,"row":0}]}
$vpRange2_ob	{"area":"ViewProArea","ranges":[{"column":5,"row":5}]}
$vpRange3_ob	{"area":"ViewProArea","ranges":[{"column":0,"row":0},{"column":5,"row":5}]}

As shown in the example above the range object is not too complex and can easily be manually created and manipulated from scratch.

Named Ranges and Formulas

With the ranges, it is possible to save them to the 4D View Pro document itself. This is done by using named ranges. A range can be named and stored into the document itself using the ***VP ADD RANGE NAME*** command. The command takes a range object for the first parameter and a unique name as the second parameter and saves the passed range into the document with the name as its unique identifier. If the name is already used the old range will be replaced with the new one.

A named range can then be accessed without needing to rebuild the range if the document is saved and closed and then reopened at another time. The command ***VP Get names*** can be used to list the ranges in the document currently displayed in the 4D View Pro area passed as the first parameter returning a collection of the named range objects which can be used. Another way to access a named range is to use the ***VP Name*** command. The command takes in the 4D View Pro area as the first parameter and the known name of the range as the second parameter returning the named range object.

In a similar concept to the named ranges, formulas can also be named. To create a named formula the command ***VP ADD FORMULA NAME*** can be used passing in the 4D View Pro area as the first parameter, the formula as the third parameter, the name as the third parameter. A list of a named formula can then be obtained using the ***VP Get formula by name*** command passing in the View Pro area as the first parameter and the name as the second returning the formula in an object with a “formula” property with the value set to the string representation of the formula. A range can also be represented as a formula using the notation of a spreadsheet.

Prior to 4Dv17 R4, there were not any native commands to interact with the contents of a 4D View Pro document. Many new native commands from the 4D View Pro component provides the ability to manipulate the contents of a 4D View Pro document entirely through code.

Programmatically Modify 4D View Pro Document Data

With the new commands introduced in 4Dv17 R4, there are several commands that will allow the programmatic manipulation of the contents of a 4D View Pro document. The commands are simple and straight forward to use with a self-defining naming convention. The commands will take in a range object as the first parameter and then the value as the second parameter to apply to all cells in the range object. Some of the commands will take a third parameter to format the values, such as dates with their multiple formats (long, short, ISO, etc.).

Below is a list of the more simple and straight forward commands. All, except for the Boolean, allow display formatting.

- ***VP SET BOOLEAN VALUE***
- ***VP SET NUM VALUE***
- ***VP SET TEXT VALUE***
- ***VP SET DATE VALUE***
- ***VP SET TIME VALUE***
- ***VP SET DATE TIME VALUE***

There are a few more commands which are a bit less simple. The ***VP SET FORMULA VALUE*** allows a formula to be added to the cell. It is a simple command to use with the knowledge of spreadsheet formulas. A more advanced feature of 4D View Pro is the ability to use a 4D project method as a formula. The formula should return a value and will need to be authorized using the **SET ALLOWED METHODS** command.

The ***VP SET VALUE*** allows for an implementation of a generic setter allowing any data type to be applied. This is done by passing in an object as the second parameter. The object has three relevant properties. The "value" property is the main property to use which allows the generic feature by accepting any value of any data type except for time. The "time" property accepts time values in seconds, this is separated from the "value" property to allow the choice of applying a date-time value instead of being limited to only one by defining a date in "value" and a "time". The last possible property is "format" which the resulting value to be visually formatted.

The ***VP SET FIELD VALUE*** allows a pointer to a table field to be added as the value however, the table and fields must be in the virtual structure. To add a table and field to the virtual structure the **SET TABLE TITLES** and **SET FIELD**

TITLES command must be used passing the optional * parameter at the end to make sure that the titles are visible in the formula editor. After doing so the command is as simple to use as the before mentioned ones even allowing formatting the display of the field however, it must be applicable to the field's data type.

While all of these features are already available by manually interacting with the 4D View Pro Area. It is sometimes preferable to procedurally generate documents based on data. These new features now allow the ability to do so from scratch or even from a template.

Reacting to the Contents

4D View Pro also provides two getters to react to the contents of the document. The first command is ***VP Get value***. It returns the value of the first cell (most upper left cell) of the range object passed into the first parameter and outputs an object. The returned object follows a similar schema to the object passed into ***VP SET VALUE*** with the "value" and "time" as possible properties, but "format" is left out as it is only a visual modifier. If the cell is empty, "value" is assigned a null.

The other command is ***VP Get formula*** which returns the formula in the first cell passed as the first parameter into a resulting string. If the cell does not contain a formula, an empty string is returned. These two commands allow procedural functions based on the contents of the 4D View Pro document allowing more control.

SpreadJS and 4D View Pro

While this is outside of the scope of 4D, the 4D View Pro area has a very powerful new feature. When broken down, the 4D View Pro area is a Web Area running SpreadJS, a javascript spreadsheet API. Because of this, some javascript and SpreadJS commands can be performed on a 4D View Pro area using **WA EVALUATE JAVASCRIPT**. The command can be applied by passing the View Pro Area as the first parameter and the script as the second parameter.

This feature provides many new options when working with the 4D View Pro area, if familiar with SpreadJS including those not yet available in a native command. This includes formatting of the cells' styles and the values styles, merging cells, and even more.

As mentioned before a 4D View Pro document is essentially an object data type that follows a certain structure. The structure's possibilities can be complex, but understanding the structure can allow documents to be created on the fly without the need of running a 4D View Pro area. The object of a View Pro document typically contains four properties. The properties are "dateCreate", "dateModified", "version", and "spreadJS". The date properties are not required when building a View Pro document but will take in an ISO formatted date time.

The “version” property is required and defaults to 1. The most important and complex property is “spreadJS” this is the actual data for the document and can be an extremely and lengthy object. The “spreadJS” schema follows the official spreadJS schema provided on their website:

https://help.grapecity.com/spread/SpreadJSWeb/fullschema.html

As long as the object associated with the “spreadJS” property is valid against the schema, the document will be able to properly load, with the “version” property defined.

Conclusion

This Technical Note provided some introductory information to the updated 4D View Pro’ features. 4D View Pro is an entirely new feature that is intended help provide the abilities to implement features that 4D View provided. This is due to the outdated 32-bit architecture that 4D View was built upon and the shift towards 64-bit. The feature set also expands to the other list boxes which can also be used to display data. The current feature set of 4D View Pro and 4D allows for most of the typical uses of the legacy 4D View to be implemented. List boxes can be used to display a list of data. The new 4D View Pro area can be used to display data in a spreadsheet. There are multiple ways to perform the last two main uses of the legacy 4D View of generating reports and displaying a calendar. With all of these needs met, 4D View Pro provides multiple commands to work with the document and work with the contents of the data. 4D View Pro even allows unimplemented commands to be used by looking at the object schema and manually building a document procedurally or looking at the spreadJS commands to interact with the 4D View Pro area. 4D View Pro is a very powerful feature that can be utilized in many ways.