

Programmatic Implementation of Conditional Formatting in 4D View Pro

By Shayanna Gatchalian, Technical Services Engineer, 4D Inc.

Technical Note 23-10

Table of Contents

Table of Contents.....	2
Abstract	3
Introduction.....	3
What is conditional formatting?	3
Why should you use conditional formatting?.....	5
What are the benefits of programmatically implementing conditional formats?	6
Conditional formatting through the 4D View Pro UI	6
Programmatic implementation of conditional formatting	6
How does conditional formatting work?	7
Walkthrough of manual conditional formatting.....	7
Walkthrough of 4D implementation of conditional formatting	10
Important Notes on Rule Precedence	12
Sample Database Walkthrough	14
Part 1: View Pro Area.....	15
Part 2: Conditional Formatting Components	15
“Add New Rule” Section	15
“Rule Priority” Section.....	16
“Current Rule Object Structure” Section	17
“View Pro Object Structure” Section.....	17
“Create Your Own Rule” Section	18
Conclusion.....	18

Abstract

While 4D View Pro automatically organizes database tables into a spreadsheet format, any sort of automatic cell or text styling would have to be done by the programmer. Conditional formatting is a special type of cell styling in which a cell range is styled only when a condition is satisfied; this allows for more dynamic interactions between the user and the workbook. While there are built-in View Pro commands to apply general cell styling, there are none for the case of conditional formatting. However, by closely comparing the SpreadJS and View Pro object structures, there is a way to programmatically implement conditional formatting. This technical note will delve into the benefits and inner mechanism of conditional formatting, as well as demonstrate how to implement it via 4D ORDA programming.

Introduction

The popularity and importance of good user interface/user experience design (a.k.a., UI/UX design) has especially grown in the last few years, as the development of technology has greatly shifted from being able to execute a function to optimizing that same function and making the task as easy as can be for the typical end-user. Although one may think of modern flyouts or intricate transition animations when he or she hears this phrase, the core concepts of UI/UX design can be applied to something as simple as reading a spreadsheet in 4D View Pro. In spreadsheet editors, cell styling serves the purpose of making a sheet more attractive and easier to read. Even more powerful, however, is the function of conditional formatting, where certain cells can change their appearance based on a condition, or rule, which creates a more dynamic sheet that changes according to what the reader would like to see. While conditional formatting is already a built-in component of SpreadJS, the Javascript class View Pro is based on, you can use 4D programming along with the SpreadJS object structure to programmatically create and apply your own conditional formats. This technical note will go over the significance of conditional formatting and how to manually and programmatically implement it in 4D View Pro.

What is conditional formatting?

Overall cell styling, on one hand, is the concept of applying text and cell styles to make a spreadsheet more attractive and feasible to read. For example, table column headers are commonly styled with a different background color and text formatting from the rest of the table; this way, the reader can easily discern between the field name and the column of data.

Student ID	First Name	Last Name	Grade
01123493	John	Doe	12
01184792	Hannah	Young	10
01128394	Cory	Baxter	11
01173829	Trent	DeGuzman	12

Table 1. Sample table of students, with differently formatted column headers

Conditional formatting, on the other hand, is changing the appearance of a cell or cell range based on a certain condition Oftentimes, this is meant to help the reader pick out certain cells or patterns in a specific group of cells. For example, if a cell range contains cells with negative number values, the background color of those applicable cells would be highlighted and bolded in red.

234	284	509	94
34	-374	30	728
743	349	22	-273
-684	-23	283	59

Table 2. Sample cell range with negative values conditionally formatted

Conditional formatting is a common feature in popular spreadsheet editors, such as Microsoft Excel, Google Sheets, and SpreadJS—the very Javascript library that View Pro is based on. Each application has a dedicated graphical interface that allows the user to apply conditional formatting to their spreadsheets. In View Pro, this is found under the “Styles” button in the ribbon at the top of the View Pro area (a.k.a., VP area).

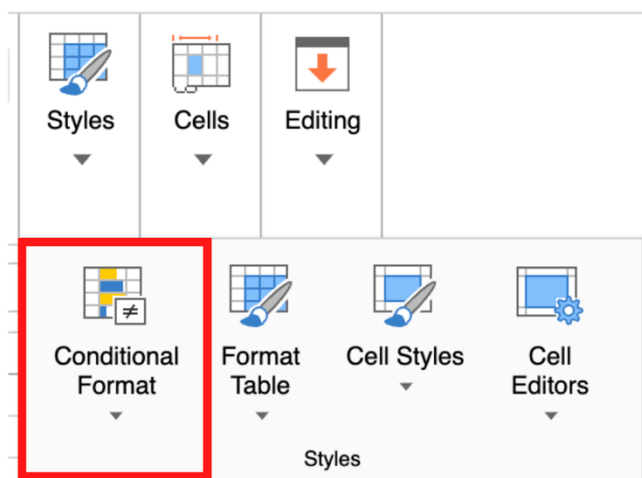


Image 1. Conditional format menu item in View Pro

With SpreadJS and View Pro in particular, conditional formatting is saved in its own “conditionalFormats” property in each of their respective object structures. SpreadJS, with the current latest version at v16, has an extensive component design complete with classes, interfaces, methods, and dedicated enumerations to allow complete control over a SpreadJS application’s design and functionality. View Pro limits 4D programmers with its smaller list of integrated View Pro commands (a.k.a., VP commands), which as of v19R8 do not include any dedicated commands conditional formatting. However, if you get to know the similar structures of the SpreadJS and View Pro objects (a.k.a., VP objects), you can programmatically implement conditional formatting in 4D View Pro using ORDA concepts.

Why should you use conditional formatting?

You may ask yourself, what is the point of conditional formatting? The data is written there on the spreadsheet and general styling is enough to make the sheet nice and easy to read; what more should I have to do? As mentioned earlier, conditional formatting makes it easier for the reader to discern certain points of data and/or patterns out of a whole spreadsheet. To better understand this, take the time to do the exercises below.

Here, you are given a sample table in the 4D View Pro area, showcasing 10 records, where each one shows the quarterly revenues of a unique small business in the year of 2017. Count how many of these values are less than \$50,000.

Company	Q1	Q2	Q3	Q4
Switch	\$60,233	\$52,932	\$42,039	\$49,029
GoGetter	\$47,423	\$24,733	\$72,278	\$55,918
TwoFriends	\$64,586	\$83,407	\$56,974	\$12,895
Kaisha	\$98,200	\$99,285	\$59,981	\$27,384
WholsIt	\$47,423	\$24,733	\$72,278	\$55,918
Dynasty	\$84,647	\$60,161	\$27,227	\$74,179
KoalaTea	\$62,378	\$85,984	\$85,566	\$16,048
Bud	\$87,893	\$48,022	\$24,779	\$10,045
OKPOP	\$84,647	\$60,161	\$27,227	\$74,179
Wakanda	\$62,378	\$85,984	\$85,566	\$16,048

Table 3. Quarterly revenues of 10 small businesses in 2017 (no conditional formatting)

How many values did you count? Now, observe the *same* table with conditional formatting applied. Each highlighted cell represents a number value that is less than 50,000. Count how many values there are now.

Company	Q1	Q2	Q3	Q4
Switch	\$60,233	\$52,932	\$42,039	\$49,029
GoGetter	\$47,423	\$24,733	\$72,278	\$55,918
TwoFriends	\$64,586	\$83,407	\$56,974	\$12,895
Kaisha	\$98,200	\$99,285	\$59,981	\$27,384
WholsIt	\$47,423	\$24,733	\$72,278	\$55,918
Dynasty	\$84,647	\$60,161	\$27,227	\$74,179

KoalaTea	\$62,378	\$85,984	\$85,566	\$16,048
Bud	\$87,893	\$48,022	\$24,779	\$10,045
OKPOP	\$84,647	\$60,161	\$27,227	\$74,179
Wakanda	\$62,378	\$85,984	\$85,566	\$16,048

Table 4. Quarterly revenues of 10 small businesses in 2017 (with conditional formatting)

Compare the time you took to complete each exercise. Was the task easier to complete the second time around? Did you get two different values after both exercises? More importantly, are your answers correct? In both tables, the number of values less than \$50,000 is 15. If you got the right answer the first time, congratulations! If not, it is okay. Either way, this goes to show how good design drastically changes a user's experience. You can look at this exercise as a smaller example of what happens in the real world when working with data, where both time and accuracy are key.

What are the benefits of programmatically implementing conditional formats?

Conditional formatting is already a built-in feature, so it is easy to wonder why one should take the extra step of learning how to implement it programmatically (from here on, the topic will be discussed in the context of 4D View Pro). Below are key differences between “manual” conditional formatting done through the View Pro interface and its programmatic counterpart.

Conditional formatting through the 4D View Pro UI

- Conditional formatting done through the View Pro UI **only applies to the current sheet of the entire workbook**; this means that if you would like to apply the same set of conditional formats to multiple pages, you would have to individually configure it for each sheet.
- Once conditional formats are applied to a sheet, they are directly tied to the View Pro object. There is **no way to inherently save the format outside of the VP object** to use on other workbooks.
- Any conditional formatting done through this method is **limited to what is provided in the View Pro interface** and can only be configured within the View Pro area.

Programmatic implementation of conditional formatting

- With this method, you can create an algorithm to **automatically apply the same conditional formatting rule(s) to multiple pages** in a workbook (more information on rules will be discussed in the next section).
- You can then **save these rules outside of the VP object** for cases where the same rules would apply to different workbooks, especially ones with the same structure but different data.

- Lastly, programmatic implementation allows the developer to **create custom UI components that tailors to specific conditional formatting needs**. This is particularly useful in cases where the View Pro ribbon/toolbar is disabled to limit what the end-user can edit on the spreadsheet.

As you can see, programmatic implementation of conditional formatting provides a way for a more robust and customized application. Although it is more work for the developer, streamlined implementation of conditional formatting would ultimately provide a better experience for the end-user viewing and manipulating the spreadsheet.

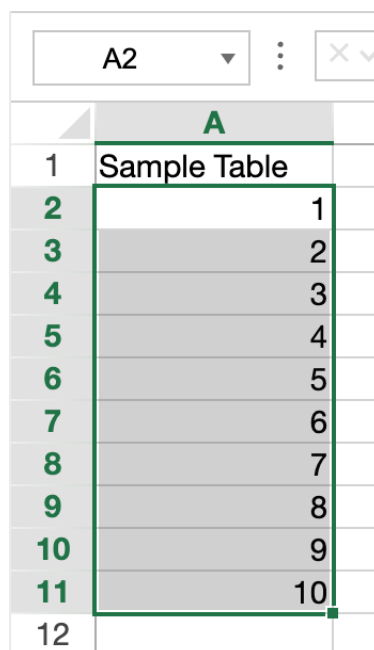
How does conditional formatting work?

Now that the key concepts and benefits of conditional formatting have been delved into, it is time to go over the two ways to apply conditional formatting: the “manual” procedure via the View Pro interface and programmatically with 4D code.

Walkthrough of manual conditional formatting

Creating a conditional formatting rule with the View Pro interface is very intuitive. You can follow the steps below as a general guide in doing so.

1. Select the cell range you would like to apply a new conditional format rule to.



	A
1	Sample Table
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10
12	

Image 2. Selection of cells to apply conditional formatting to

2. In the ribbon at the top of the VP area, click on the “Styles” button, then the “Conditional Formats” button. A context menu will appear with a list of the common types of conditional formatting rules.

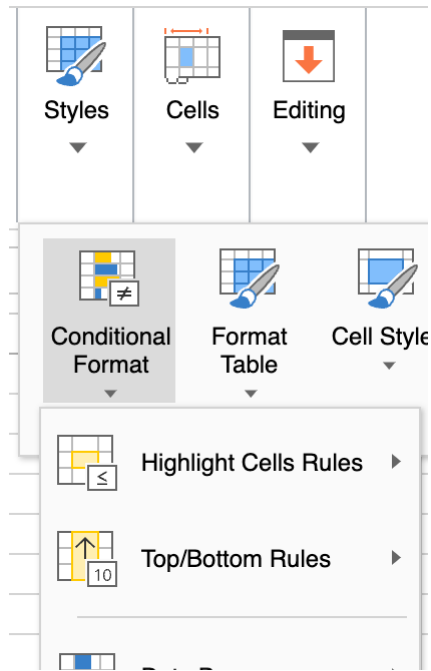


Image 3. Conditional format context menu

3. You can use any of these options, but you may access an extensive list of rule types by clicking on “New Rule...”.

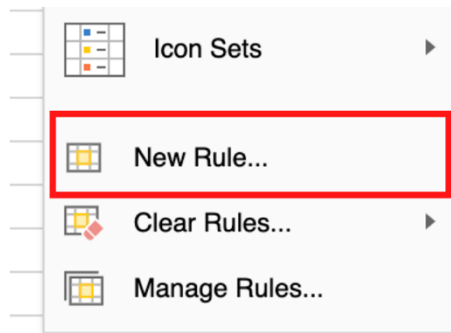


Image 4. “New Rule...” option under conditional format context menu

4. A dialog window should appear, where you can choose the rule type in the “Select a Rule Type” section and edit the rule’s properties in the “Edit the Rule Description” section. More information on rule types can be found in the “The Different Types of Conditional Formatting Rules” document attached with this technical note.

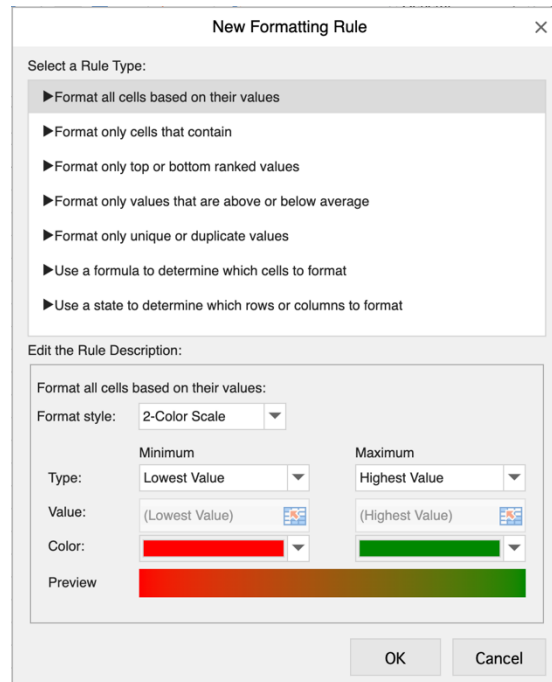


Image 5. “New Formatting Rule” dialog window

5. Click “OK” and the conditional format rule will be applied to the sheet. You can view currently applied conditional formatting by going to Styles > Conditional Formats > Manage Rules... Here you can edit or delete currently existing rules.

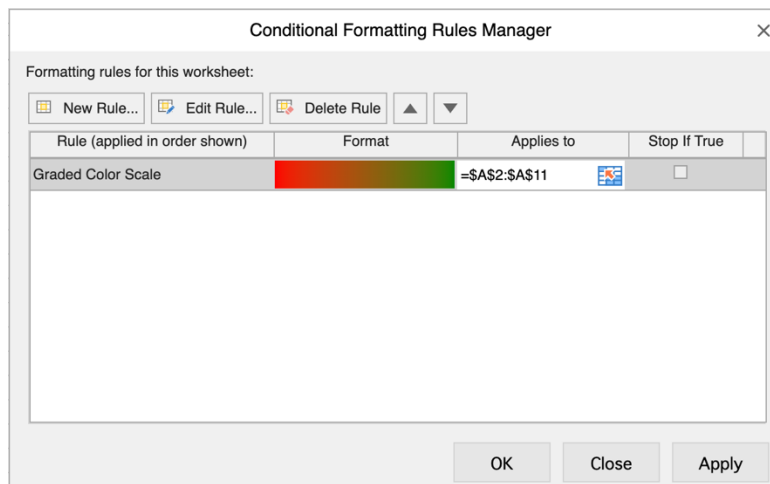


Image 6. “Conditional Formatting Rules Manager” dialog window

Walkthrough of 4D implementation of conditional formatting

Although 4D does not offer built-in 4D View Pro commands to manipulate conditional formats, a workaround can be used by manipulating the View Pro object with ORDA and modeling its conditional format property after that of SpreadJS. The following algorithm and explanation of the VP object structure can be used as a model for 4D implementation of conditional formatting.

First, the VP object must be transferred into the context of 4D by using the **VP Export to object** command. The object itself stores the entire content of the VP document and is comprised of five properties:

- version
- dateCreation
- dateModified
- meta
- SpreadJS

4D View Pro Object Documentation: <https://doc4d.github.io/docs/ViewPro/configuring#4d-view-pro-object>

Assume that we are working with a new document where no conditional formats have been applied yet. Here, we want to examine the “SpreadJS”, which is of the object type. It contains several levels of properties, but the “conditionalFormats” property will be found under “sheets” > [the name of the sheet you would like to apply a conditional format to]; the default “Sheet1” page will be used for this example.

Expression	Value
▼ \$viewProObj	{ "version":1,"dateCreation":...tType":"vt:i4","value":"2"}...
dateCreation	"2023-04-28T22:19:10.695Z"
dateModified	"2023-05-12T17:38:11Z"
▼ spreadJS	{ "version":"15.2.5","docPro...Type":"vt:i4","value":"2"}...
customList	[]
docProps	{ "docPropsCore":{"lastModi...ype":"vt:i4","value":"2"}...
iterativeCalculation	False
iterativeCalculationMaximumChange	0.001
iterativeCalculationMaximumIterations	100
namedStyles	[{"backColor":null,"foreColo...p":null}, {"backColor":null...
pivotCaches	{}
sheetCount	1
▼ sheets	{ "Sheet1":{"name":"Sheet1",...ecoration":0,"imeMode":1...
Sheet1	{ "name":"Sheet1","isSelecte...ecoration":0,"imeMode":1...
sheetTabCount	0
tabStripRatio	0.6
version	"15.2.5"
version	1

Image 7. Structure of the View Pro object until the “sheets” property

The “conditionalFormat” property is an optional element that will only appear if there are any defined on this sheet; since we are working with a new document, the attribute is absent from the object. At this point, you can create your own conditional format rule to apply to the “Sheet1” object.

Each conditional format is defined as a “rule”, which is comprised of three main parts:

- The **condition** to be met (this can be thought of as the “If” condition)
- The **styling** of the cell range
- The **cell ranges** the rule is applied to

In SpreadJS, each rule type is assigned an enumeration value (i.e., a constant) to help define which properties are needed to correctly execute the conditional format. There are 14 rule types:

- Average Rule
- Cell Value Rule
- Column State Rule
- Data Bar Rule
- Date Occurring Rule
- Duplicate Rule
- Formula Rule
- Icon Set Rule
- Row State Rule
- Specific Text Rule
- Three Scale Rule
- Top 10 Rule
- Two Scale Rule
- Unique Rule

SpreadJS Rule Type Enumeration Values:

<https://www.grapecity.com/spreadjs/api/enums/GC.Spread.Sheets.ConditionalFormatting.RuleType>

SpreadJS “ConditionalFormats” Class:

<https://www.grapecity.com/spreadjs/api/classes/GC.Spread.Sheets.ConditionalFormatting.ConditionalFormats#class-conditionalformats>

In the “Conditional Formatting Rule Types” document, you can find the corresponding enumeration for the 14 rule types as well as a detailed list of properties needed to construct each type of rule object. The condition component can be thought of as the if-statement that must be satisfied for the formatting to take place. Lastly, the cell range corresponds to the group or groups of cells in which the rule is applied to. Most, but not all, rules also require the style property that describes how the cell range should look when the condition is satisfied.

After constructing the rule object, it can be saved anywhere outside of the VP document in different forms (e.g., a variable, as part of a record in the 4D database, a JSON object, etc.) and can be applied to other workbooks for consistency and time-saving purposes.

Multiple rules can then be gathered into a collection of “rules”. This collection is what you will attach to the “conditionalFormats” property. Once it is saved into the VP object, the conditional formatting will stick with the document until it is deleted.

Expression	Value
<ul style="list-style-type: none"> \$viewProObj <ul style="list-style-type: none"> dateCreation <ul style="list-style-type: none"> dateModified spreadJS <ul style="list-style-type: none"> customList docProps <ul style="list-style-type: none"> iterativeCalculation iterativeCalculationMaximumChange iterativeCalculationMaximumIterations namedStyles pivotCaches sheetCount sheets <ul style="list-style-type: none"> Sheet1 <ul style="list-style-type: none"> activeRow autoMergeRangeInfos cellStates colHeaderData columnCount columnOutlines columns conditionalFormats <ul style="list-style-type: none"> rules <ul style="list-style-type: none"> length rules[0] data frozenTrailingColumnStickToEdge frozenTrailingRowStickToEdge 	<pre> {"version":1,"dateCreation...ype":"vt:i4","value":"2"}... "2023-04-28T22:19:10.695Z" "2023-05-12T17:38:11Z" {"version":"15.2.5","docPr...ype":"vt:i4","value":"2"}... [] {"docPropsCore":{"lastMo...pe":"vt:i4","value":"2"}}... False 0.001 100 [{"backColor":null,"foreCol...":null},{"backColor":null... {} 1 {"Sheet1":{"name":"Sheet...ecoration":0,"imeMode":1... {"name":"Sheet1","isSele...ecoration":0,"imeMode":1... 1 [] {} {"defaultDataNode":{"style":{"themeFont":"Body"}}} 19 {"items":[]} [{"size":111},{"size":106},{"size":126},{"size":101}] {"rules":[{"ruleType":10,"ra...e":2,"maxColor":"#080"}]} [{"ruleType":10,"ranges":[{"...pe":2,"maxColor":"#080"}]} 1 {"ruleType":10,"ranges":[{"...pe":2,"maxColor":"#080"}]} {"dataTable":{"0":{"0":{"va...":null,"diagonalUp":null}}... True True </pre>

Image 8. Structure of the View Pro object until the “rules” property

The object can then be imported back into the VP area for further data manipulation in the context of the View Pro; this is done using the **VP Import from object** command. It must be noted that **any changes done to the VP object in this process are volatile**, meaning that they will not be saved onto the VP document until the file itself is saved on the disk.

Important Notes on Rule Precedence

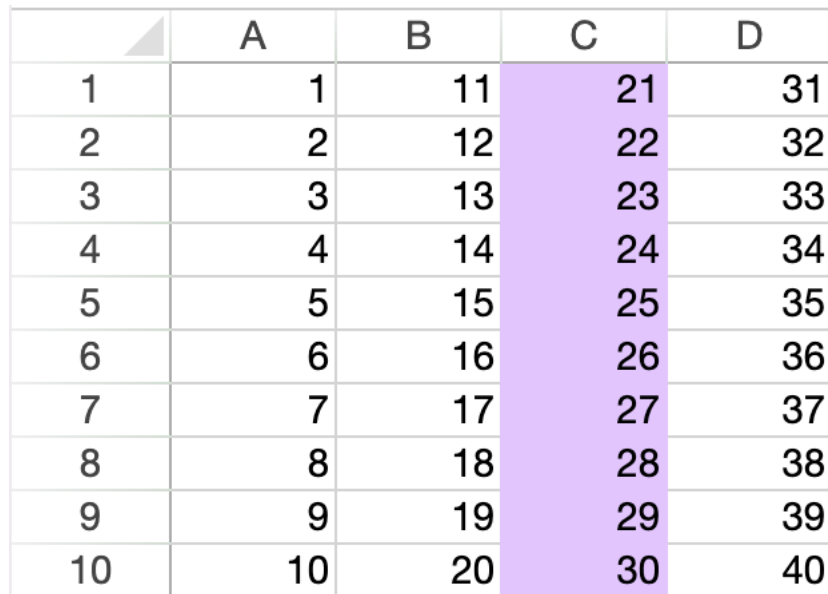
When discussing rule precedence in spreadsheet editors, there are two contexts that you must be mindful of. The first of which is regarding the **overall order of conditional formatting rules** for the entire sheet. The first item in the rules collection has least precedence, while the last item has most precedence. Because rules are appended to the end of a collection, the most recently added rule would automatically have the most precedence. The order of precedence can be manipulated simply by changing the order of

rule objects in the rules collection of the “ConditionalFormats” property in the View Pro object.

SpreadJS Conditional Format Rule Priority documentation:

<https://www.grapecity.com/spreadjs/docs/features/condformat#priority-of-conditional-formats>

The second context has to do with the situation where the **ranges of two or more rules overlap and multiple rules are applied to the same cells**. This can happen when the ranges are the same or when the two ranges have partial overlap. In the images below, blue and red cells represent an individual cell range, while purple cells represent overlap between two cell ranges.



	A	B	C	D
1	1	11	21	31
2	2	12	22	32
3	3	13	23	33
4	4	14	24	34
5	5	15	25	35
6	6	16	26	36
7	7	17	27	37
8	8	18	28	38
9	9	19	29	39
10	10	20	30	40

Image 9. Visual representation of complete overlap of cell ranges

	A	B	C	D
1	1	11	21	31
2	2	12	22	32
3	3	13	23	33
4	4	14	24	34
5	5	15	25	35
6	6	16	26	36
7	7	17	27	37
8	8	18	28	38
9	9	19	29	39
10	10	20	30	40

Image 10. Visual representation of a partial cell overlap of cell ranges

In this case, the precedence of these rules is defined by the “priority” property of the rule object. Rule objects with no “priority” property will have highest priority on the overlapping ranges, whereas subsequent rules will then have a priority of 2, 3, 4, and so on.

SpreadJS Priority Property Documentation:

<https://www.grapecity.com/spreadjs/api/classes/GC.Spread.Sheets.ConditionalFormatting.ConditionRuleBase#priority>

Sample Database Walkthrough

This technical note comes with a sample 4D database attached, called “ConditionalFormatsDemo”. Upon opening the project, you are first welcomed with a splash screen, where if you click on the “Open Demo” button, the demo application will pop up. The application interface can be split into two parts.

Part 1: View Pro Area

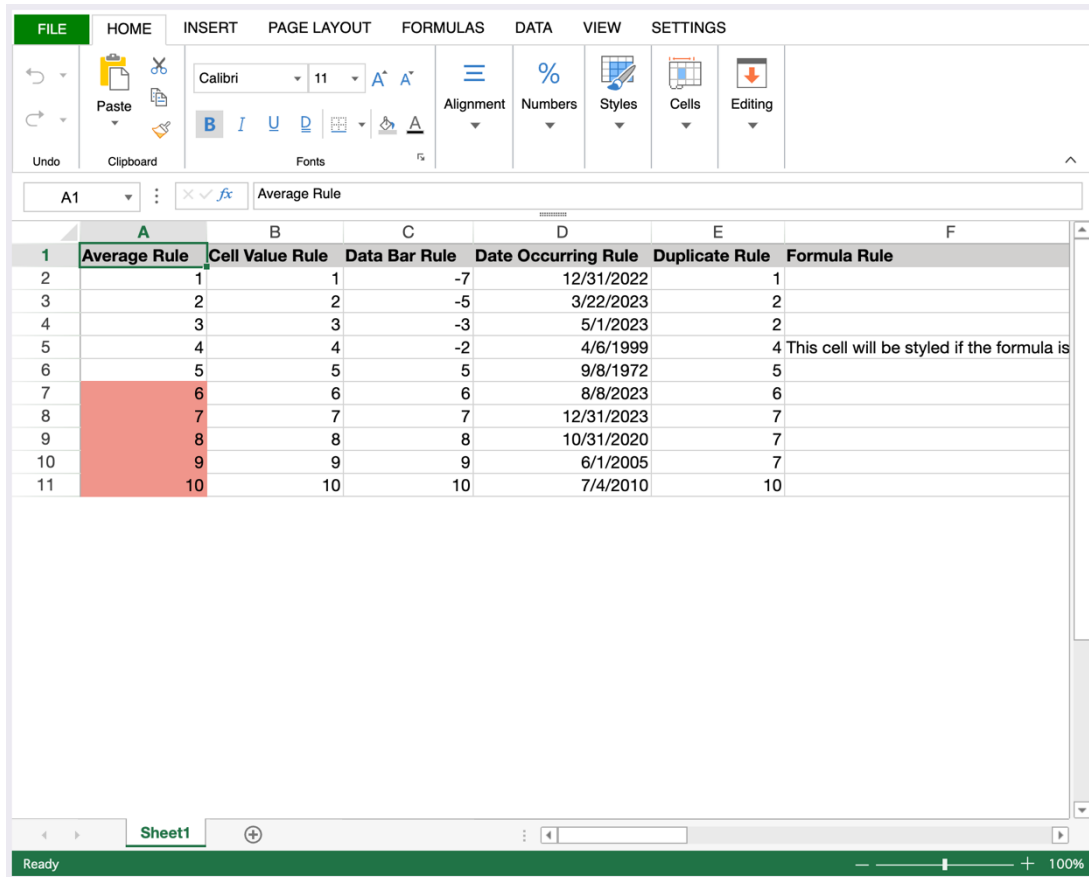


Image 11. The View Pro Area of the sample database

On the left-hand side is the View Pro area, where the spreadsheet editor is defaulted with the toolbar interface, which does not include conditional formatting options like the ribbon does. In the spreadsheet area is a template where each column is dedicated to showcasing each rule type or variation on the implementation of a rule type.

Part 2: Conditional Formatting Components

On the right are form components to manipulate conditional formatting rules on the spreadsheet.

“Add New Rule” Section

Add New Rule

Average Rule

The average rule applies cell styling according to either the average value or standard deviation of the cell range.

Apply

Image 11. The “Add New Rule” section of the sample database

The “Add New Rule” section displays a dropdown list of 16 different implementations of the available rule types provided by SpreadJS. Each rule type features a brief description of the condition needed to be satisfied for the cell styling to take place. Once you choose a rule type, you can Apply the selected rule by clicking on the “Apply” button.

“Rule Priority” Section

Rule Priority

You can change a rule's priority using the "↑" and "↓" buttons below

Rule Type	Ranges	Style Property?
Average Rule	A2:A12	✓

▲
▼

Delete Rule

Image 12. The “Rule Priority” section of the sample database

The “Rule Priority” section features a list box showcasing each of the currently applied conditional formatting rules. The displayed fields in this list box are derived from the View Pro object but are saved as extra properties in the Form.rulesCol form data; these additional properties do not affect the View Pro object because they are disregarded when imported into the View Pro area. The list box is comprised of three columns,

“Rule Type”, “Ranges”, and “Style Property?”, to help the user easily identify which rule is applied to which cell ranges. The “Rule Type” column translates the integer value of the “ruleType” property into the actual name of the rule type. The “Ranges” column iterates through each of the range objects in the “ranges” collection and formats each cell range as the letter-number combination for a cell’s index (e.g., “A1:A10”). Lastly, the “Style Property?” column indicates whether a certain rule contains a style object. You can change the current rule’s priority using the up and down buttons under the list box.

“Current Rule Object Structure” Section

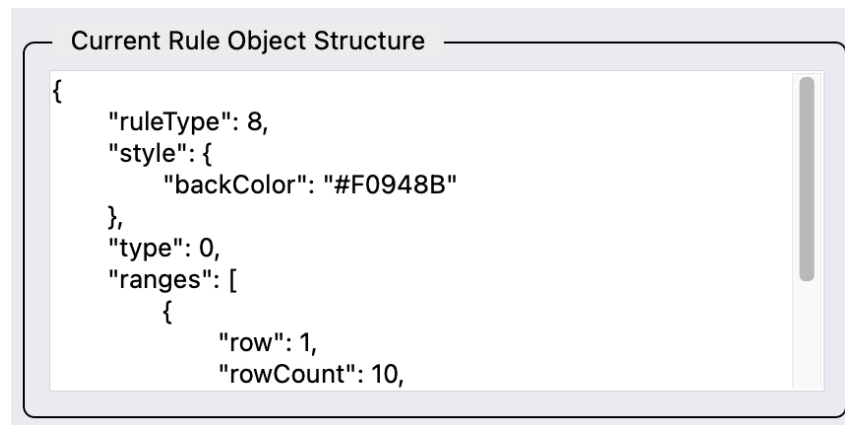


Image 13. The “Current Rule Object Structure” section of the sample database

The section is very straightforward as it takes the currently selected rule object in the collection and displays it as a prettified JSON string in a text input box. The displayed string is a result of the “currentRuleStructure” method, where it retrieves the selected rule object in the View Pro object, rather than from the Form.rulesCol form data. This is because the Form.rulesCol rule objects contain extra properties that do not adhere to the View Pro structure (see “Rule Priority” section).

“View Pro Object Structure” Section

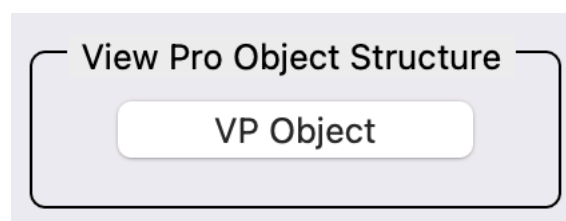


Image 14. The “View Pro Object Structure” section of the sample database

This section only contains a button that opens a debugger to allow the user to view the entire structure of the View Pro object. This is helpful in getting familiarized with where the “conditionalFormats” property belongs.

“Create Your Own Rule” Section



Image 15. The “Create Your Own Rule” section of the sample database

The “Create Your Own Rule” section offer you a chance to implement your own conditional format rule for a more hands-on learning experience, by implementing the “My Rule” button’s object method. As is, the button would throw logic errors since it only contains skeleton code. You would simply need to follow the comments in the object method and use your newfound knowledge and resources to construct your own rule object. Then, you can test it on the sample sheet with its own dedicated column labeled “My Rule” at the end of the sheet.

Conclusion

With the sample database and the “Conditional Formatting Rule Types” document included, this entire technical note package serves as an all-in-one resource for conditional formatting in 4D View Pro. This document went over the basics of conditional formatting, its benefits in the context of better serving the user’s viewing experience, and how to apply conditional formatting rules via the View Pro interface, as well as through 4D implementation. In a world taken over by big data and analytics, it is easy to be overwhelmed with so much information being passed around at unprecedented speeds. That is why it is important that this knowledge be accessible and displayed in a way that is feasible to understand. Although 4D View Pro does not handle the more intensive processes just as the standalone program does, it can serve as the bridge between data and the common person.