



# Table Views Guide

8.5.0 Release

---

Copyright © 2024 OneStream Software LLC. All rights reserved.

Any warranty with respect to the software or its functionality will be expressly given in the Subscription License Agreement or Software License and Services Agreement between OneStream and the warrantee. This document does not itself constitute a representation or warranty with respect to the software or any related matter.

OneStream Software, OneStream, Extensible Dimensionality and the OneStream logo are trademarks of OneStream Software LLC in the United States and other countries. Microsoft, Microsoft Azure, Microsoft Office, Windows, Windows Server, Excel, Internet Information Services, Windows Communication Foundation and SQL Server are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. DevExpress is a registered trademark of Developer Express, Inc. Cisco is a registered trademark of Cisco Systems, Inc. Intel is a trademark of Intel Corporation. AMD64 is a trademark of Advanced Micro Devices, Inc. Other names may be trademarks of their respective owners.

# Table of Contents

Table Views Spreadsheet and Excel Add-In .....	1
Overview .....	2
Technical Features and Setup .....	3
Restrictions .....	3
Table View Sizing .....	5
Table Views Spreadsheet/Excel Ribbon Button .....	5
Table View Business Rules .....	8
Spreadsheet Function Types .....	8
Processing Order .....	9
Using Parameters .....	9
Can Modify Data .....	10
Table View Conditions .....	11
Table View Sources .....	11
Table View Business Rule Example .....	12
GetTableView Function Type .....	12
Incorporating Parameters .....	22
Using XFTV Named Ranges .....	22

**Table of Contents**

---

Security ..... 26

Summary ..... 29

Sample Table View Rules File ..... 30

# Table Views Spreadsheet and Excel Add-In

The primary purpose of Table Views is to provide a method for accessing or updating relational data. This data is presented in a dashboard or inside the Excel Add-In. The use of Table Views enables the designer to work in a more flexible environment to design a form or data collection tool.

Table Views are not alternatives to other tools, such as the SQL Table Editor or Grid Viewer, Dashboard Components.

Key Use:

- Designed to collect records from relational tables, or other sources
- Present the information in the Spreadsheet format
- Utilize client-side functionality, found in the Spreadsheet tool, such as calculations and pick-list validation lists
- Table View Business Rules can be designed to manage the column field records, such as updates, inserts and deletes.

Design Considerations:

- The current functionality is designed to update records in target tables
- Controlling elements must be designed into the Table View Business Rule by the creator to ensure data integrity, security and performance

Table View Size Considerations:

- Table Views depends upon the number of rows and row content
- Paging is not supported. Therefore, all rows and content must be returned
- Performance testing and design expectations is to support approximately 8000 KB of data per Table View.

## Overview

A Table View definition for the Windows Application Spreadsheet Tool or Excel Add-In is defined in a Business Rule. The Administrator designing the rule can define the rows and columns which should be returned to the worksheet from the source table presented in the **Table View**.

The Table View Business Rule can collect data from multiple data sources. For example, a single worksheet can display a Table View which collects data from two or more sources.

The Administrator has full control over the write back “save” process through Business Rules. When designing the Table View Business Rule, the BRAPI Authorization functions should be designed into the Business Rule to control access to the viewing or modifying the data. This can be applied to the entire table or to specific cells. A workbook can contain multiple Table Views. These can be on the same worksheet or across worksheet pages.

A single Business Rule file can be used to define multiple Table Views by calling the Business Rule argument, **TableViewName**. Additionally, a single named range can be used to manage table data cells within the Spreadsheet and Excel Add-In using user defined named ranges (**XFTV\_\***).

# Technical Features and Setup

This section will review the various functional elements of the **Table Views** feature. The design of Table Views involves having a thorough understanding of the source and target tables to be viewed or modified. The **Administrator** developing the Table View will also be required to understand the requirements needed for the final Spreadsheet form to design the Business Rule at its most granular level. This will allow the Business Rules to be designed to the most restrictive level which will maximize security and gain the highest performance.

## Restrictions

Table Views should never read or write to OneStream Application controlling tables, such as Data Tables, Cube Tables or Log Tables.

- AppProperty\*
- Attachment\*
- Audit\*
- Data\*
- CalcStatus\*
- Certify\*
- Confirm\*
- Cube\*
- Dashboard\*
- DataAttachment\*

## Technical Features and Setup

---

- DataCellDetail\*
- DataEntry\*
- DataMgmt\*
- DataRecord\*
- DataUnit\*
- Dim\*
- FileContents\*
- FileInfo\*
- Folder\*
- Form\*
- FxRate\*
- ICMatchStatus\*
- Journal\*
- Member\*
- Parser\*
- Relationship\*
- SecRoles\*
- Stage\*
- System\*
- Taskflow\*
- Time\*



- Workflow\*

## Table View Sizing

The output interface to the Table View Business Rule is the OneStream Windows Application Spreadsheet and Excel Add-In.

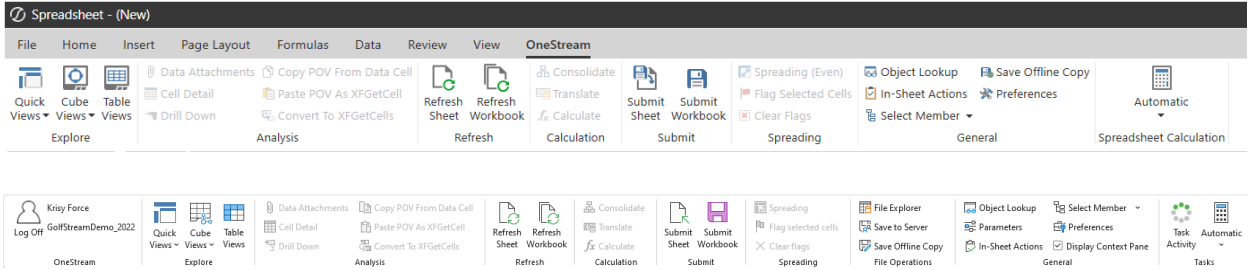
Table Views should not be considered as a replacement for other Dashboard tools used with database tables, such as the **SQL Table Editor** or the **Grid View** components which support very large tables.

The Spreadsheet tool and Excel Add-In does not have a paging function to manage very large data sets. Therefore, careful testing is recommended to verify the size and performance of the records being managed with Table Views.

A significant impact on the performance of Table Views is the **cell content**. Along with the physical number of rows, the content contained in the cells can dramatically affect performance. The cell content is the key factor on the impact of the ultimate size on disk.

## Table Views Spreadsheet/Excel Ribbon Button

**Table Views** is a OneStream Windows Application Spreadsheet and Excel Add-In feature used to assign a Spreadsheet Business Rule to a worksheet. All Table Views are derived through the definition of a Business Rule, and only Administrators have the rights to create Business Rules.



## Technical Features and Setup

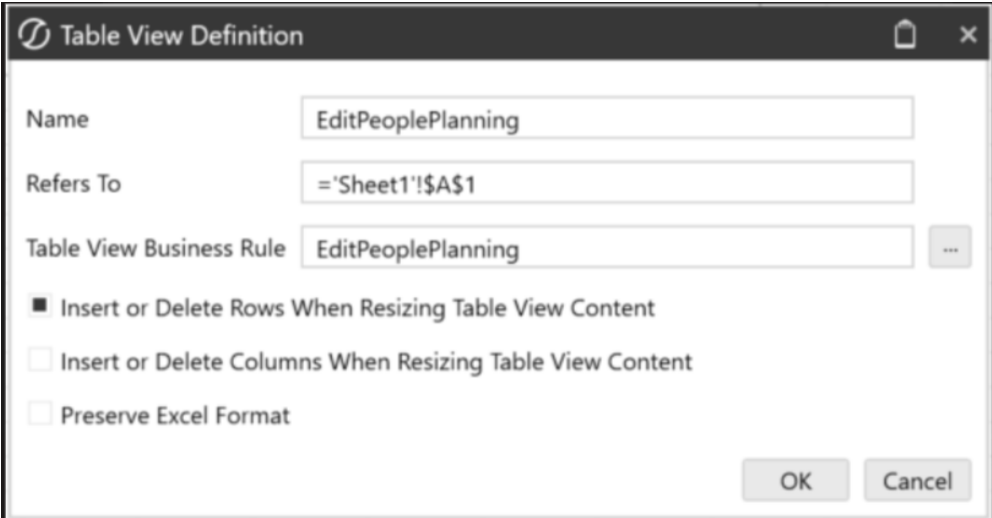
---

1. Open the OneStream Windows Application and select **Tools/Spreadsheet** or Open your **Excel Add-In**.
2. Select an available cell to begin the Table View range.
3. From the **OneStream** tool bar, choose the **Table Views** button.

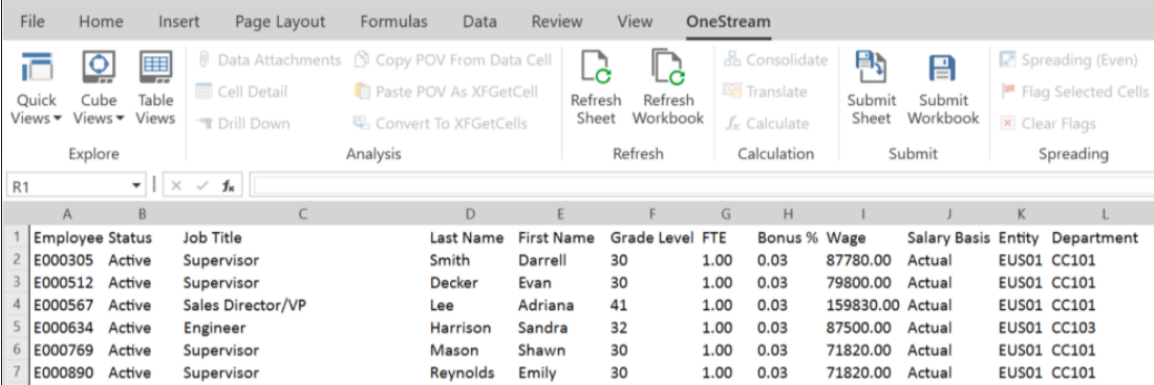


4. Choose the **Add** button. Selecting ellipsis button from the **Table View Business Rule** field allows browsing the available Business Rules. The selection will automatically assign the **Name** and **Refers To** cell intersection. Only **Spreadsheet** type Business Rules will render as a Table View.

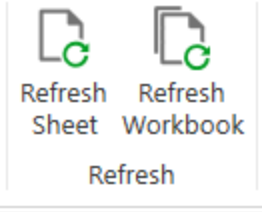
# Technical Features and Setup



5. The Table View will render in the worksheet and is associated with a named range.



6. Choosing the Refresh options will retrieve the most current results from the source table.



# Table View Business Rules

Access to **Table Views** in Spreadsheet and Excel Add-In is limited to the **Spreadsheet Business Rule Type**. The purpose of the Business Rule is to establish the source data records to be displayed. The ability to save a record or field within a record is also completely defined within the Business Rule. The Table View Business Rules also support Parameters to enable the resulting Worksheet to be included in complex Dashboards.

## Spreadsheet Function Types

- **GetCustomSubstVarsInUse** Used to define the interaction with OneStream Dashboard Parameters
- **GetTableView** Used to define the source(s) for the Table View.
- **SaveTableView** This function defines the table or cell intersection that should be written to a target database table

```
Select Case args.FunctionType
    Case Is = SpreadsheetFunctionType.Unknown
    Case Is = SpreadsheetFunctionType.GetCustomSubstVarsInUse
        Return Nothing
    Case Is = SpreadsheetFunctionType.GetTableView
        Return GetEmployeeDetails(si, args.CustSubstVarsAlreadyResolved)
    Case Is = SpreadsheetFunctionType.SaveTableView
        Return UpdateEmployeeDepartment(si, args.TableView)
End Select
```

### Processing Order

The Spreadsheet Function Types are designed to manage the processes within a common Dashboard environment.

1. **GetCustomSubstitutionVariables** is executed first.
  - a. If the defined Parameter is contained within the Dashboard, the selection will act as a bound parameter and will be passed into the business rule.
  - b. If the defined Parameter is not contained within the Dashboard, it will run/prompt the user.
  - c. Additional conditional Parameters will be executed. The Spreadsheet Business Rules can conditionally execute additional Parameters, based on the results of resolved Parameters.
2. Once all the Parameters are resolved, the **GetTableView** function will be processed. This section will generate the results in the Table View. The Table View will also be evaluated to determine if there will be any writable conditions. If there are no writable conditions, which is the default, any refresh of the Spreadsheet/Table View will restart at the GetCustomSubstitutionVariables function.
3. If the GetTableView is flagged as a writable table, the **SaveTableView** process will be executed, writing back only the elements specifically defined in the Business Rule.

### Using Parameters

The **GetCustomSubstitutionVariables** function is used to incorporate Parameters into the Table View. Any parameters required are passed in as a list within the Function Type. If

## Technical Features and Setup

---

the Parameter is not included in the supporting Dashboard and resolved, for example as a Combo box, the Parameter will be executed in the Table View to be resolved.

```
Dim list As New List(Of String)
list.Add("Param_SelectEntity")
```

Additional Parameters can be included in the Table View to act as a nested, conditional Parameter using the **custSubstVarsAlreadyResolved** function. This enables a resolved Parameter to be evaluated to trigger additional Parameters to execute. The **custSubstVarsAlreadyResolved** can conditionally evaluate all resolved parameters to determine subsequent parameters to be executed.

```
If custSubstVarsAlreadyResolved.ContainsKey("Param_SelectEntity")
    list.Add("Param_SelectDepartment")
End If
```

## Can Modify Data

All Table Views will default to “read only”. The Table View condition for **CanModifyData** must be set to True to allow write-back capability. The **CanModifyData** object is set in the **GetTableView** Function Type. It is only required if any write-back is required based on the current Table View. The **True** condition will enable objects to be passed, and enabled, in to the **SaveTableView** Function Type. When refreshing a Table View, the **SaveTableView** Function Type will not be executed unless the **CanModifyData** property is set to True.

```
Dim tableView As New TableView()  
tableView.CanModifyData = True
```

## Table View Conditions

A single Spreadsheet Business Rule can contain multiple Table View definitions. The Table View Name can be called using the **Args.TableViewName** to allow conditionally calling rule functions.

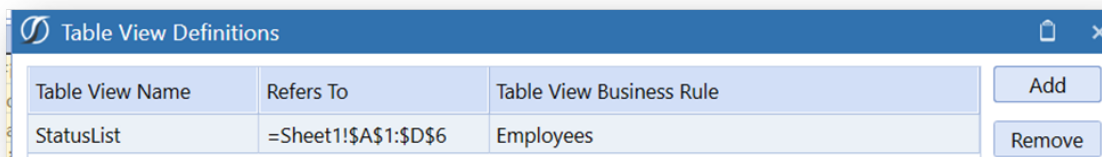


Table View Name	Refers To	Table View Business Rule	Add
StatusList	=Sheet1!\$A\$1:\$D\$6	Employees	Remove

```
Case Is = SpreadsheetFunctionType.GetTableView  
  If args.TableViewName.Equals("Employees")  
    Return GetAllEmployeeDetails(si, args.CustSubstVarsAlreadyResolved, False)  
  End If
```

## Table View Sources

Table View Business Rules can collect a variety of data records as a source. Typically, a source is defined as a table from a database. It is not limited to a single table but can collect records from multiple tables. The Table View Business Rule designer can define the source essentially as any data accessible to the Spreadsheet Business Rules.

Similarly, the **SaveTableView** rules can be defined to any target accessible by the Business Rules.

## Table View Business Rule Example

This is an example only for the purpose of outlining the basic elements of a Table View Business Rule. By default, a Table View is “read only”. A Spreadsheet Business Rule can be defined to return a complete table. Always consider the size and content of the table as it may impact performance. Elements that can impact performance, such as exceeding the ability to render the Table View, are the total number of rows as well as the content within the records.

## GetTableView Function Type

### Database Connection

Create connections to sources, such as a database table using business rules.

```
Dim sql As New Text.StringBuilder
sql.AppendLine("Select * ")
sql.AppendLine("From Employees ")

'Create and fill the data table
Dim dt As DataTable = Nothing
Using dbConnApp As DbConnInfo = BRApi.Database.CreateApplicationDbConnInfo(si)
    dt = BRApi.Database.ExecuteSql(dbConnApp, sql.ToString, False)
    If Not dt Is Nothing Then dt.TableName = "ContentList"
End Using
```

### Determine if the Table View Requires Write-Back

If the Table View must write-back to a target database or table, the **CanModifyData** property must be set to **True**.



## Technical Features and Setup

---

```
Dim tableView As New TableView()  
tableView.CanModifyData = True
```

### Define the Table View Columns

Table columns can be returned for the entire table, or as distinct items. When columns are defined, they can be returned to the Table View using an alias description as part of a Header section.

```
'Create Columns on Table View  
'Create a column header  
Dim tableViewRowHeader As New TableViewRow()  
'Return all columns from the data table  
For Each dataColumn As DataColumn In dt.Columns  
    Dim column As New TableViewColumn()  
    column.Name = dataColumn.ColumnName  
    'rename the table column  
    If column.Name.Equals("Employee_Status") Then  
        column.Value = "Status"  
    Else  
        'return the table column name  
        column.Value = dataColumn.ColumnName  
    End If  
    column.IsHeader = True  
    tableView.Columns.Add(column)  
    'generate column headers based on the column name  
    tableViewRowHeader.Items.Add(column.Name, column)  
Next dataColumn  
tableView.Rows.Add(tableViewRowHeader)
```

Create a nested, parameter-driven combo box in a Table View column by adding the following code to your business rules:

```
1 | TableViewColumn tableViewColumn1 = oTableView.CreateColumn("ParamName1", "Column1",  
    true, "Default.[pf8_1322_delimited_h_path_1]", true);
```

### Returning Rows to the Table View

Each row cell is evaluated from the data table columns. The designer has full control over the display of the content of the table using Business Rule functions. In the example below, the presentation of the results will vary by column, by user using the BR API Security Authorization function.

```
'Create Data Row Records
For Each dataRow As DataRow In dt.Rows
    Dim tableViewRow As New TableViewRow()
        For Each tableViewColumn As TableViewColumn In tableView.Columns

            Dim column As New TableViewColumn()
            Dim columnValue As String = ""
            column.Name = tableViewColumn.Name
            columnValue = dataRow.Item(tableViewColumn.Name)
            'Condition to limit view of results to only Administrators
            If column.Name.Equals("SSN") Then
                If Not BrApi.Security.Authorization.IsUserInAdminGroup(si) Then
                    columnValue = "XXX-XX-" + columnValue.Substring(7, 4)
                End If
            End If
            column.Value = columnValue
            column.IsHeader = False

            tableViewRow.Items.Add(tableViewColumn.Name, column)
        Next TableViewColumn
    tableView.Rows.Add(tableViewRow)
Next dataRow

Return tableView
```

### Security Filtering Results

	A	B	C	D	E	F
1	Id	First	Last	Department	Status	SSN
2	1	John	Smith	Dev	Active	XXX-XX-1234
3	2	Jane	Smith	Dev	Active	XXX-XX-4567

## Technical Features and Setup

---

	A	B	C	D	E	F
1	Id	First	Last	Department	Status	SSN
2	1	John	Smith	Dev	Active	000-00-1234
3	2	Jane	Smith	Dev	Active	000-00-4567

### Add New Records

Add new records to a table by assigning a specific range of editable rows at the bottom of the Table View, which can be used by rules to commit the records into a table. Format the background area with a fill color to visually indicate the area is enabled for adding new records.

Use the Insert Rows feature to insert empty rows into a table and change the background color.

- **CanModifyData:** Set to True to False to determine if the table can contain empty rows.
- **NumberOfEmptyRowsToAdd:** Set the number of empty rows to add.
- **EmptyRowsBackgroundColor:** Set the color of the background.

```
Dim tableView As New TableView()  
tableView.CanModifyData = True  
tableView.NumberOfEmptyRowsToAdd = 10  
tableView.EmptyRowsBackgroundColor = XFColors.Blue
```

The following example shows the business rule applied to the table.

## Technical Features and Setup

---

	A	B	C	D	E	F	
1	Id	First Name	Last Name	Department			
2	1	John	Smith	Dev			
3	2	Jane	Smith	Dev			
4	3	Lisa	Kron	HR			
5	4	Henrik	Ibsen	Sales			
6	5	Joyce	Oates	Sales			
7	6	Sara	Flynn	Sales			
8	7	John	Smith	Dev			
9	8	Jane	Smith	Dev			
10							
11							
12							
13							
14							
15							
16							
17							

### Data Type Object for Column Fields

The `DataType` object allows the designer to define the Column Field as Text or Numeric. This object references the current `XFDataType` object. However, not all `XFDataType` properties are valid for Table Views. Only **Int16**, **Int32**, **Int64**, **Float**, **Double**, **Decimal**, and **Text** are valid.

If you do not specify a data type, it will default to **Text**.

#### 'Add Columns to the Table View

```
tableView.Columns.Add(CreateTableViewColumn("Id", "Id", True))
tableView.Columns.Add(CreateTableViewColumn("First", "First Name", True))
tableView.Columns.Add(CreateTableViewColumn("Last", "Last Name", True))
tableView.Columns.Add(CreateTableViewColumn("Department", "Department", True))
```

```
Dim salaryColumn = CreateTableViewColumn("Salary", "Salary", True)
salaryColumn.DataType = XFDataType.Decimal
tableView.Columns.Add(salaryColumn)
```

## Technical Features and Setup

---

In the example below, the Salary column is rendering the Table View Column fields as numeric values to accurately reflect their nature and will support Spreadsheet based calculations.

	A	B	C	D	E
1	Id	First Name	Last Name	Department	Salary
2	1	John	Smith	HR	80000.11
3	2	Jane	Smith	DEV	70000.12
4	3	Lisa	Kron	DEV	50000.11
5	4	Henrik	Ibsen	HR	50000
6	5	Joyce	Oates	HR	50000.11
7	6	Test	One	DEV	50000
8	7	Test	Two	DEV	50000
9	8	Test	Three	DEV	50000
10	9	Test	Four	HR	45123.23
11	10	Test	Five	DEV	41345.2345
12	11	Test	Six	HR	903.123
13					
14					
15					
16					
17					

### Enable Status Column

The Table View Business Rule can create a dedicated status column. In the example below, it is My Status column. Use this to classify records for use in conditional business rule logic to drive behaviors.

In this example, the business rule can define members for a drop-down list defined as Delete, Archive, and Inactive. The designer creates business rules to perform actions based on the status of the records, such as delete, or archiving to another table.

Use the Enable Status Column option to manage records for your table.

- **statusColumnEnabled:** creates a status column in the table view when set to True.
- **statusColumnName:** string defines the name of the column. If left blank, the default name “XFTV\_Status” will be assigned.

# Technical Features and Setup

- **statusColumnIndex**: zero-based integer identifies the column where the status is created. A value above the actual number of columns will assign the Status as the last Column. A negative number will assign the Status column as the first column.
- **statusColumnValues**: creates a list of members to select as a validation in the Status column. It is a hidden range at the top of the Table View. If left blank, no list or validation will automatically be created in the Status column, it will need to be created manually by the designer.

```
'If multiple SetCustomStatusColumn statements are set on the same table view, it will take the last statement.  
tableView.EnableStatusColumn(True, statusColumnName, 4, "DELETE,ARCHIVE,INACTIVE")
```

In the screenshot below, notice the Delete, Archive, Inactive, which is entered in the business rule.

	A	B	C	D	E	F	G	H	I	J
4	Id	First Name	Last Name	Department	MyStatus					
5	1	John	Smith	Dev						
6	2	Jane	Smith	Dev	DELETE					
7	3	Lisa	Kron	HR	ARCHIVE					
8	4	Henrik	Ibsen	Sales	INACTIVE					
9	5	Joyce	Oates	Sales						
10	6	Sara	Flynn	Sales						
11	7	John	Smith	Dev						
12	8	Jane	Smith	Dev						
13										
14										
15										
16										
17										
18										
19										
20										
21										
22										
23										
24										

### Write Back

If the **GetTableView** Function Type is modified to set the Table View property **CanModifyData** as True, the **SaveTableView** Function will execute. This section is used by the designer to define which records should write back to the target. The target table does not have to be the same as the source table.

Control conditions should be designed into the write-back rules for efficiency and performance. For example, Member Functions, such as **IsDirty()** can be incorporated to write only the modified members within the writeable records.

#### Member Functions

- **IsDirty**– Condition Check if the item has been modified
- **IsHeader**– Member record status as a Header record.
- **Name** – Member label of the data table. Will not reference an alias label.
- **OriginalValue**– Condition reflects last stored value prior to the Table View refresh
- **Value**– Reflects the current value present on the Spreadsheet Table View. This can be a changed, unsaved value.

```
Dim sql As String
Dim Id As String
Id = ""
Dim department As String
department = ""
Using dbConnApp As DbConnInfo = BRApi.Database.CreateApplicationDbConnInfo(si)
    For Each tableViewRow As TableViewRow In tableView.Rows
        If tableViewRow.IsHeader = False
            For Each tableViewColumn As TableViewColumn In tableView.Columns

                If tableViewColumn.Name = "Id"
                    Id = tableViewRow.Item(tableViewColumn.Name).Value
                End If

                If tableViewColumn.Name = "Department"
                    Dim tableViewCellDepartment As TableViewColumn
                    tableViewCellDepartment = tableViewRow.Item(tableViewColumn.Name)

                    If tableViewCellDepartment.IsDirty() Then
                        department = tableViewCellDepartment.Value
                    Else
                        department = ""
                    End If
                End If
            End If

            Next tableViewColumn
            If Not String.IsNullOrEmpty(department) Then
                sql = "Update Employees Set Department = '" & department & "' Where Id = " & Id & " "
                BRApi.Database.ExecuteSql(dbConnApp, sql, False)
            End If
        End If
    Next tableViewRow
End Using
```

### Create Table View From Data Table

You can create a Table View from Data Table using the Table View

**PopulateFromDataTable** function. The new function has two additional Boolean properties to include a Header Row and to utilize the Data Table's Data Type. The function is able to utilize any Data Table, including those from Dashboard Data Adapters using the GetAdoDataSetForAdapter function.

Properties:

- tableView.PopulateFromDataTable(data Table , Include Header Row, Include Data Types)



### Column Format Object

The ColumnFormat Object allows the Table View Designer to format the content area of a column, while excluding the Column Header for use as a separately formattable column header using the HeaderFormat object.

#### **tableView.Columns(1).ColumnFormat.ColumnWidth = 15**

- BackgroundColor
- ColumnWidth
- FontFamily
- FontSize
- IsBold
- IsItalic
- IsUnderlined
- TextColor
- NumDecimals
- AsPercentage

### Header Format Object

The use of the HeaderFormat Object requires the PopulateFromDataTable to include a header or a scripted data table to define a TableViewRow as IsHeader=True. This function allows a column headers to be formatted as a row using all the formatting options except NumDecimals and AsPercentage.

#### **tableView.HeaderFormat.BackgroundColor = XFColors.Navy**

# Incorporating Parameters

**CAUTION:** The OneStream Parameters to be bound, or used, in the Spreadsheet Table View are defined in the **GetCustomSubstVarsInUse** Function Type. The Parameters can be resolved as a component within a Dashboard, or they can be an element of the Table View. Once resolved, the Parameter is passed to the **GetTableView** Function Type.

```
Private Function GetCustomSubstVarsInUse(ByVal si As SessionInfo, ByVal custSubstVarsAlreadyResolved As Dictionary(Of String, String)) As List(Of String)
    Try
        Dim list As New List(Of String)
        list.Add("LastName")

        If custSubstVarsAlreadyResolved.ContainsKey("LastName")
            list.Add("Department")
        End If

        Return list
    Catch ex As Exception
        Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
    End Try
End Function

Private Function GetEmployeeDetails(ByVal si As SessionInfo, ByVal custSubstVarsAlreadyResolved As Dictionary(Of String, String)) As Table
    Try
        Dim sql As New Text.StringBuilder
        sql.AppendLine("Select Id, First, Last, Department ")
        sql.AppendLine("From Employees ")

        If custSubstVarsAlreadyResolved.ContainsKey("LastName")
            sql.AppendLine("Where Last = '" & custSubstVarsAlreadyResolved("LastName") & "' ")
        End If

        If custSubstVarsAlreadyResolved.ContainsKey("Department")
            sql.AppendLine(" and Department = '" & custSubstVarsAlreadyResolved("Department") & "' ")
        End If
    End Try
End Function
```

## Using XFTV Named Ranges

The purpose of creating a Spreadsheet using the “XFTV” named range is to manage data cells with read and write functionality to a Table View. This eliminates much of the work related to creating dashboards which may require multiple text boxes, labels, combo boxes, business rules and other controls to manage data across a table.

The XFTV Named Range can be used to link a field to a Table View. For example, a list of members may be used in a drop-down list. The selected item would then be used to

## Technical Features and Setup

---

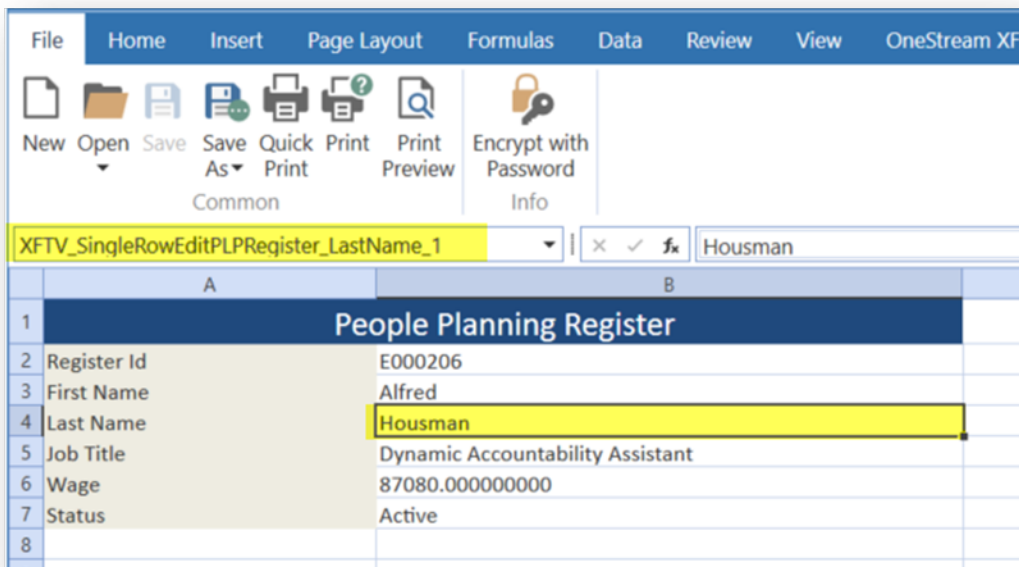
write back to a required field in a Table View, which would ultimately write to a target data source.

A cell used as a Table View reference must be prefixed with **XFTV\_** to pass into a Table View. The structure of the named range is “Prefix\_Table View Name\_Column Name\_Row Number”. The row number position is a zero-based index.

### Example

Sheet1 is designed as an interface or form based on records sourced from a table.

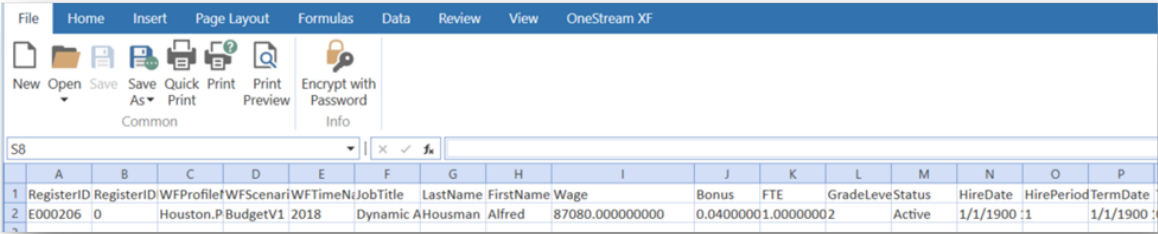
The data cell items are organized on the primary sheet with each being set as a XFTV named range referencing Sheet2, which is the core Table View.



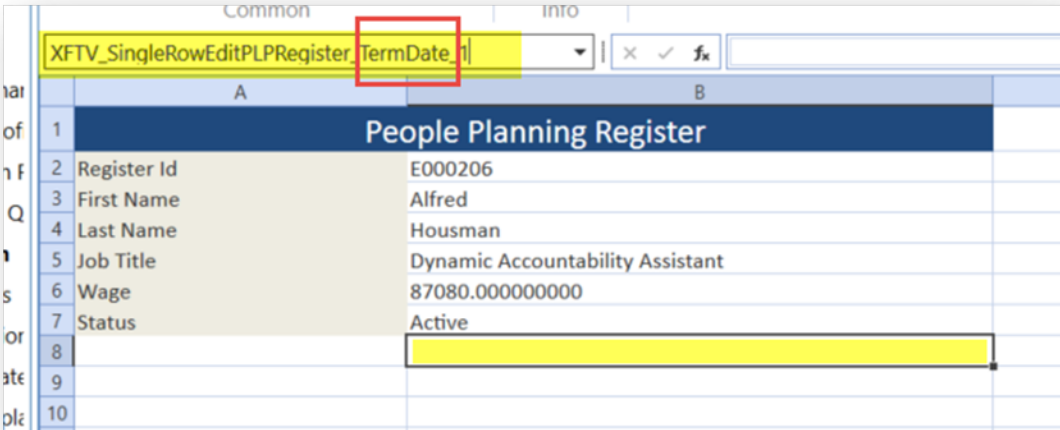
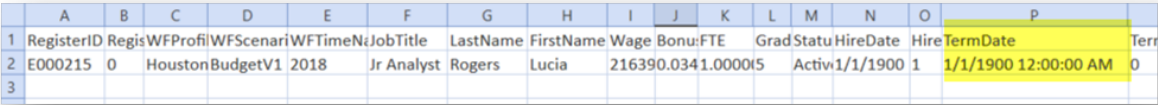
Sheet 2 is a Spreadsheet as defined by a Table Rule Business Rule

The Table View is added to the sheet and corresponds to the XFTV range definition on Sheet 1. The XFTV named ranges associate their value to the Table View for read or write processing dependent upon the Table View rule construction.

# Technical Features and Setup



Modifying the Sheet1 “form” for an additional field simply requires adding a named range. As an example, the “form” may require an additional field which may be found as a referenced validation or from the source table view. For example, the “TermDate” field may be required. Selecting the cell and adding the syntax for the XFTV named range, for the appropriate field, will incorporate the results into the sheet.



# Technical Features and Setup

---

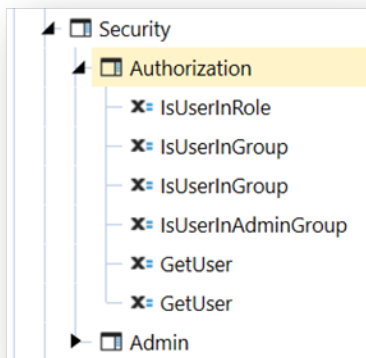
The data will automatically refresh from the defined source. If defined as a write-back field, changes to the cell can be written back to a target table using the “submit” function.

	A	B
1	<b>People Planning Register</b>	
2	Register Id	E000215
3	First Name	Lucia
4	Last Name	Rogers
5	Job Title	Jr Analyst
6	Wage	216390.000000000
7	Status	Active
8	Term Date	1/1/1900 12:00:00 AM
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

# Security

Security is controlled by the Business Rule Developer in three ways. It is very important that the business rule designer/author consider data security when creating table views. The session info object within the rule can be used to restrict/grant data access for the current user. Second, the writeback functionality will also be controlled within the business rule to the user population allowed to perform the writeback, as well as the granular level elements which may be modified. Lastly, the Table View Business Rule itself should be secured for viewing or access outside of the defined dashboard.

Data level, or Table level, security is incorporated within the Business Rule script. Various BRAPI functions can be conditionally included in the script to control the read and write functionality each user will encounter when presented with the Table View. Using Table View Name arguments in the Business Rule, rather than relying on the default Business Rule Name, will also add an additional level of security for related to the tables.



Business Rule level security should also be utilized to restrict access to those who can edit and modify the underlying Table View Business Rule. This can be done by using

## Security

---

**Business Rule Encryption**, which requires specific a user security role. Business Rule Encryption applies password protection to any Business Rule it is applied to.



**Encrypt Business Rule**

1. Password must be between 8 and 16 characters
2. Must contain at least one uppercase letter
3. Must contain at least one lowercase letter
4. Must contain at least one number
5. Must contain at least one special character.  
Valid special characters are limited to "@&!\$#%^\*()"
6. No spaces allowed

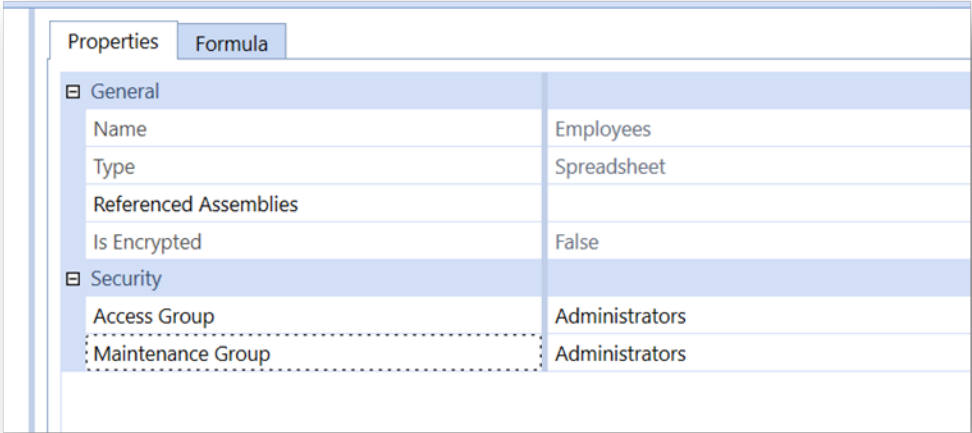
Password

Reenter New Password

OK Cancel

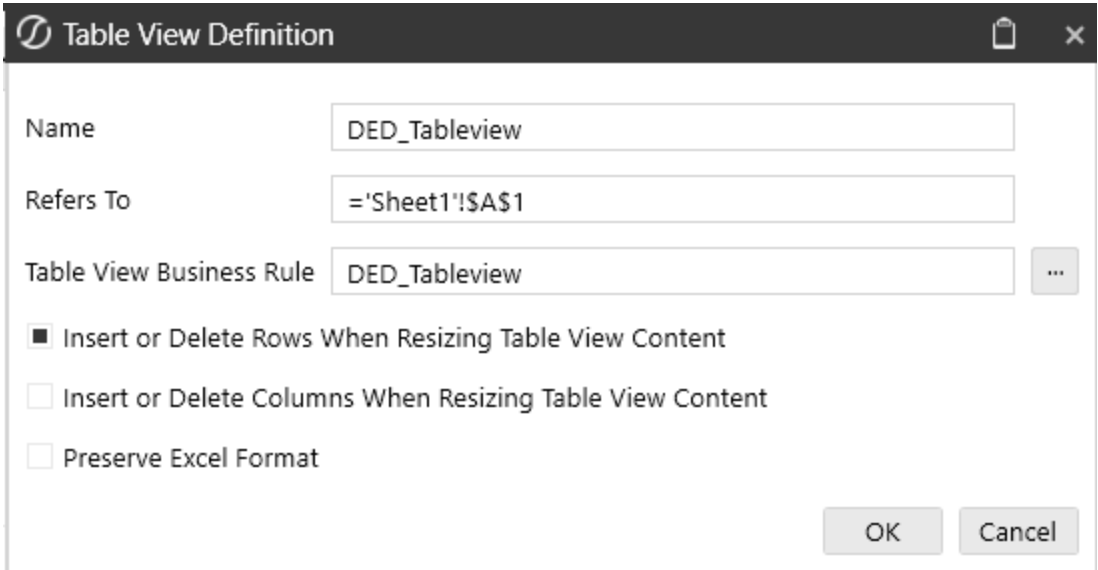
Additionally, the Business Rules for **Table Views** are stored in the **Spreadsheet** category. To control access to user's access to retrieving the Table Views in their Application Spreadsheet, the **Access Group** on each rule should exclude any user who is not a designer.

# Security



The Table View function should be called using a condition for the Spreadsheet Table View Name. The will control all Table View functionality by a defined name, rather than through the business rule alone.

```
Case Is = SpreadsheetFunctionType.GetTableView  
If args.TableViewName = "DED_TableView"  
Return GetPLPRegisterUsingRegisterID(si, args.Cust)  
End If
```

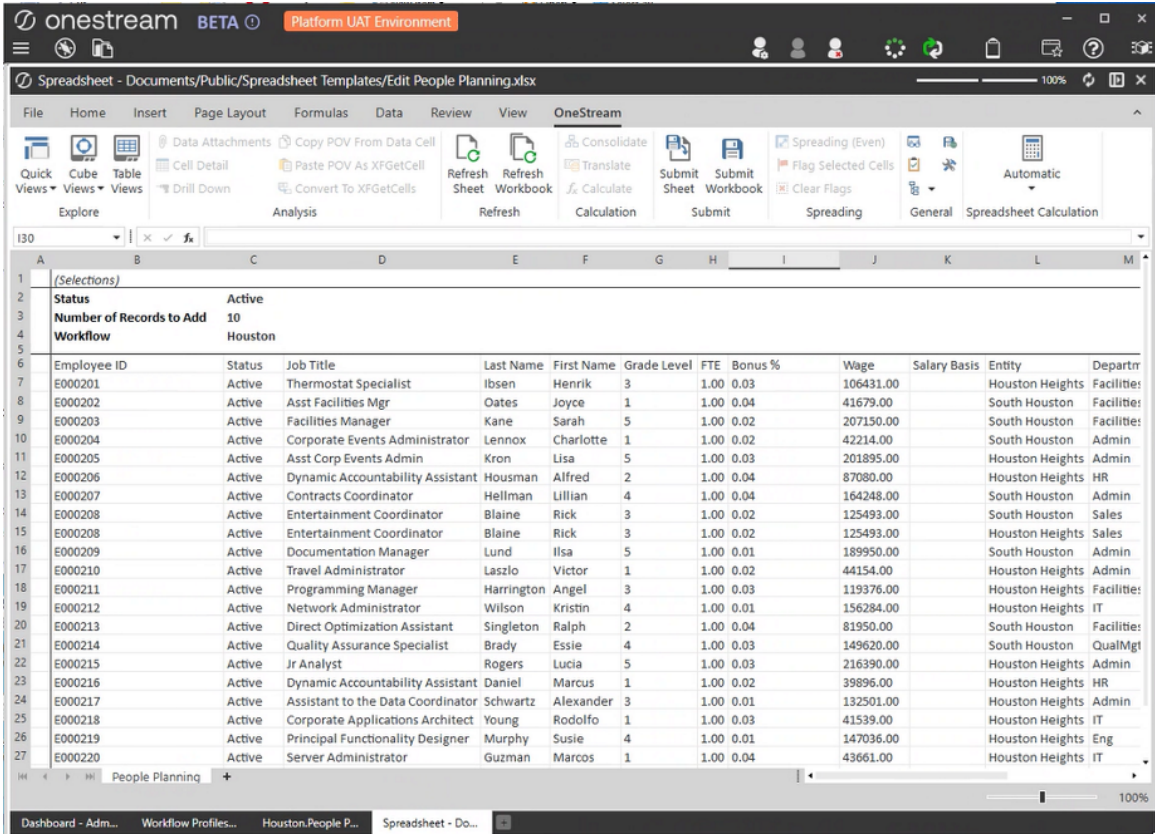




# Summary

# Summary

The **Table Views** feature is intended to provide a flexible solution for Dashboard “form” development when an update to a table is required. This business rule-based solution can manage records from a variety of sources, as well as control the target and granularity of the write-back records. This feature fully supports Dashboard based Parameters as well as additional levels of Table View based parameters to build rich Spreadsheet based Dashboard interfaces.



# Sample Table View Rules File

```
Namespace OneStream.BusinessRule.Spreadsheet.TableViewSample

Public Class MainClass

Public Function Main(ByVal si As SessionInfo, ByVal globals As
BRGlobals, ByVal api As Object,

        ByVal args As SpreadsheetArgs) As Object

        Try

        Select Case args.FunctionType

                Case Is = SpreadsheetFunctionType.Unknown

                Case Is =

SpreadsheetFunctionType.GetCustomSubstVarsInUse

                Return GetCustomSubstVarsInUse(si,
args.CustSubstVarsAlreadyResolved)

                Case Is =

SpreadsheetFunctionType.GetTableView
```

## Sample Table View Rules File

---

```
        'The same business rule can support  
multiple Table Views.
```

```
        If args.TableViewName.Equals  
("MyTableViewName")
```

```
            Return GetMyTableView(si,  
args.CustSubstVarsAlreadyResolved)
```

```
        End If
```

```
        Case Is =  
SpreadsheetFunctionType.SaveTableView
```

```
            SaveMyTableView(si, args.TableView)
```

```
        End Select
```

```
        Return Nothing
```

```
    Catch ex As Exception
```

```
        Throw ErrorHandler.LogWrite(si, New XFException  
(si, ex))
```

```
    End Try
```

```
End Function
```

```
Private Function GetCustomSubstVarsInUse(ByVal si As SessionInfo,  
ByVal custSubstVarsAlreadyResolved
```

## Sample Table View Rules File

---

```
As Dictionary(Of String, String)) As List(Of String)

    Try

        'You will be prompted for the value of these
variables if they have not been resolved.

        Dim list As New List(Of String)

        list.Add("MyTableViewParameterName")

        Return list

    Catch ex As Exception

        Throw ErrorHandler.LogWrite(si, New XFException
(si, ex))

    End Try

End Function

Private Function GetMyTableView(ByVal si As SessionInfo, ByVal
custSubstVarsAlreadyResolved

As Dictionary(Of String, String)) As TableView

    Try

        Dim sql As New Text.StringBuilder

        sql.AppendLine("Select * from MyTable")
```

## Sample Table View Rules File

---

'You can use substitution variables that have been resolved within the query.

```
        If custSubstVarsAlreadyResolved.ContainsKey
("MyTableViewParameterName")

sql.AppendLine("Where MyFilterColumn = '" &
custSubstVarsAlreadyResolved("MyTableViewParameterName") & "' ")

        End If
```

```
        'Create and fill the data table

        Dim dt As DataTable = Nothing

        Using dbConnApp As DbConnInfo =
BRApi.Database.CreateApplicationDbConnInfo(si)

                dt = BRApi.Database.ExecuteSql(dbConnApp,
sql.ToString, False)

                If Not dt Is Nothing Then dt.TableName =
"NoData"

        End Using
```

```
        'Create the Table View object

        Dim tableView As New TableView()
```

## Sample Table View Rules File

---

'This allows the Table View data to be updated.  
This is set to False by default.

```
tableView.CanModifyData = True
```

'Create Columns on Table View using the Data  
Table columns.

'Adding a header row to the Table View is  
optional

```
Dim tableViewRowHeader As New TableViewRow()
```

```
For Each dataColumn As DataColumn In dt.Columns
```

```
    'You can conditionally hide a column
```

```
    'If Not Convert.ToString  
(dataColumn.ColumnName).Equals("MyColumnToHide")
```

```
        Dim column As New TableViewColumn()
```

```
        column.Name = dataColumn.ColumnName
```

```
        column.Value = dataColumn.ColumnName
```

```
        column.IsHeader = True
```

```
        tableView.Columns.Add(column)
```

```
        tableViewRowHeader.Items.Add(column.Name,  
column)
```

```
    'End If
```

## Sample Table View Rules File

---

```
Next dataColumn

tableView.Rows.Add(tableViewRowHeader)

'Create Column Data Rows

For Each dataRow As DataRow In dt.Rows

    Dim tableViewRow As New TableRow()

        For Each tableViewColumn As
tableViewColumn In tableView.Columns

            Dim column As New TableColumn()

            Dim columnValue As String = ""

            column.Name = tableViewColumn.Name

            columnValue = dataRow.Item

(tableViewColumn.Name)

            'You can show/hide/mask column
conditionally (e.g. based on the user group)

            If column.Name.Equals

("MySensitiveData") Then

                If Not

BrApi.Security.Authorization.IsUserInAdminGroup(si) Then

                    columnValue = "Not Available"

                End If
```

## Sample Table View Rules File

---

```

        End If

        column.Value = columnValue

        column.IsHeader = False

        tableViewRow.Items.Add
(tableViewColumn.Name, column)

        Next TableViewColumn

        tableView.Rows.Add(tableViewRow)

    Next dataRow

    Return tableView

Catch ex As Exception

    Throw ErrorHandler.LogWrite(si, New XFException
(si, ex))

End Try

End Function

Private Function SaveMyTableView(ByVal si As SessionInfo, ByVal
tableView As TableView) As Boolean

    'Add code to check if the user has permission to
write data.

    If Not tableView Is Nothing
```



## Sample Table View Rules File

---

```
        Dim sql As String = String.Empty

        Dim tableViewMyPrimaryKey As New
tableViewColumn()

        Dim tableViewMyColumnToUpdate As New
tableViewColumn()

        Using dbConnApp As DbConnInfo =
BRApi.Database.CreateApplicationDbConnInfo(si)

            For Each tableViewRow As TableViewRow In
tableView.Rows

                If tableViewRow.IsHeader = False

                    For Each tableViewColumn As
tableViewColumn In tableView.Columns

                        If tableViewColumn.Name =
" MyPrimaryKey"

                            tableViewMyPrimaryKey = tableViewRow.Item
(tableViewColumn.Name)

                                End If

                                    If tableViewColumn.Name =
" MyColumnToUpdate"

                                        tableViewMyColumnToUpdate =
tableViewRow.Item(tableViewColumn.Name)

                                            End If
```

## Sample Table View Rules File

---

```

                                Next tableViewColumn

                                'Update the column value only if
the value was changed.

                                If
tableViewMyColumnToUpdate.IsDirty()

                                'Create audit records as needed
before and after updating data.

sql = "Update MyTable Set MyColumnToUpdate = '" &
tableViewMyColumnToUpdate.Value & "' Where

                                MyPrimaryKey = " & tableViewMyPrimaryKey.Value & " "

                                BRApi.Database.ExecuteSql
(dbConnApp, sql, False)

                                End If

                                End If

                                Next tableViewRow

                                End Using

                                End If

                                Return True

                                End Function

                                End Class

End Namespace
```