

ExcelRT

Mac, Windows & Cloud

User's Guide

Version 3.1

Excel Software

www.excelsoftware.com • info@excelsoftware.com

Phone: 702-445-7645

Copyright and Trademarks

MacA&D™, MacTranslator™, WinA&D™, WinTranslator™, XojoApp™, MarketBuddy™, ClickInstall™, QuickLicense™, QuickLicense Server™, DocProtect™, WebActivation™, LicenseSupport™, AppProtect™, QLRT Xcode™, OfficeProtect™, PluginFMQLRT™, PluginXojoQLRT™, PluginProtect Photoshop™, Cloud License™, Cloud License Server™, Web License Server™, Desktop License Server™, QuickLicenseRT Linux™, MakeDongle™, Safe Activation™, ExcelRT™, ExcelCL™ and CloudRT™ are trademarks of Excel Software. Copyright 1985-2021 by Excel Software with all rights reserved. Other trademarks are property of their respective owners and used for illustrative purposes only.

This manual and software are copyrighted with all rights reserved. Under the copyright laws, this manual or the software may not be copied, in whole or part, without written consent of Excel Software, except in the normal use of the software by licensed users or to make a backup copy. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original.

Disclaimer

EXCEL SOFTWARE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THIS SOFTWARE, ITS QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE IS SOLD "AS IS", AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE.

IN NO EVENT WILL EXCEL SOFTWARE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE OR ITS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In particular, Excel Software shall have no liability for any programs or data stored in or used with Excel Software products, including the costs of recovering such programs or data.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESSED OR IMPLIED. No Excel Software dealer, agent or employee is authorized to make any modification, extension or addition to this warranty.

Contents

Chapter 1

Setup ExcelRT 1-1

Introduction	2
ExcelRT Overview	4
Design vs User Mode	5
Limits and Differences	6
Supported Platforms	7
Install on Windows	7
Install ExcelRT	7
Install ConvertExcelRT	7
Run ExcelRT or ConvertExcelRT	8
Windows DLL Update	8
Uninstall	8
Install on Mac	9
Install ExcelRT	9
Uninstall ExcelRT	9
Activation	9
Move a License	11
Restore a License	11
Quick Start	12
About This Book	13
Support Services	13

Chapter 2

Develop ExcelRT 2-1

ConvertExcelRT	2
Overview Dialog	4
ExcelRT Design Mode	5
Cell Edit	5
Design for 100% Zoom	5
Pictures	6
Tables	6
Encrypted File	7
Exceptions	7
Performance	8
Feature Differences	9
Cell Format	9
Functions	9
Range References	9
Range Reference Function	9
3D References	10
Formula Parameter Nesting	10

Calculation Precedence	10
Range Selection	10
Data Entry	10
Data Validation	11
Pictures & Form Controls	11
Background Images	11
Structured Table References	11
Text Across Cells	12
Stylized Text	12
Unicode Text	12
Recalculation Algorithm	13
Design for Performance	14
Sheet Type	15
Fonts	16
Design Mode Interface	18
User Mode Interface	19
Conversion Checklist	21

Chapter 3

Program ExcelRT 3-1

Button Actions	1
Script Commands	2
Vendor Commands	3
Server Setup	3
RegisterServer	3
Log	4
Upload	4
Email	4
Email Server	6
Storage	8
Message	9
Query	11
Cloud Sharing	13
Settings Dialog	15
Event Actions	16
External Commands	18
Clipboard Command	18
File Command	18
Create a Script	19
Variables	20
Text Editor	21
Script Line Continuation	21

Test a Script	22	Plugin Files	54
Download Plugin File	22	Google Map and Direction	57
Custom Commands	22	Pictures	58
Using a Command as a Parameter	25	Form Controls	60
Importing a Script	26	HTML Control	61
Custom Functions	27	Command Log	65
		Arrays	66
Chapter 4		Cloud License	68
Script Commands	4-1	Progress	69
Cell Data Import and Export	1	Switch Rows and Columns	70
Platform Specific	3	Sound	71
Shared Ticket Folder	4	Python	72
Custom Images	6	JSON	72
Internet Data	7	Database	75
Dialog with Button Actions	9		
External Applications	10	Chapter 5	
Clipboard and File Data	10	Deploy ExcelRT	5-1
Prompt for Data Entry	10	Generate ERT File	2
Variables	12	Standalone App	3
Prompt User	14	ExcelRT on Customer Computer	5
Cell References	15	Activate Standalone App	7
Math	16	Update Standalone App	8
Lists	16	Deploy Plugin Files	9
Email Read	18	Screen Size and Orientation	10
External App Communication	18	Purchase Button	11
Picture and PDF Files	20		
Subroutines	21	Chapter 6	
Conditionals and Loops	22	Charts	6-1
Symbols	24	Chart Types	4
CSV Read, Write and Modify	26	Chart Styles	7
Multiple CSVs	29	Chart Components	8
Fuzy Query	30	Chart Dimensions	9
String	32	Chart Data	10
Dialog	34	Build and Use Charts	10
ReportBuilder	36		
Cell Data	38	Chapter 7	
Cell Controls	39	ExcelRT Builder	7-1
Sheet Resize	42	Builder Tools	3
Feature Control	43	Sheet Add Delete	7
Credit Card and Paypal	43	Sheet Size	7
Stripe Payment	46	Erase Cell Data	8
Message Dialog	48	Cell Size & Visibility	8
Miscellaneous	49	Format Rules	8
Platform Specific Files	52	Control Add Delete	9
Splitting & Scaling Images	53		

Control Edit	10
Pictures	11
Cell Control	12
Table Add Delete	13
Cell Borders	14
Cell Validation	14
Cell Text Color	15
Cell Background	15
Cell Copy & Paste	16
Paste Cell Range	16
Sheet Filter	17
Merged Cells	17
HTML Control	18
ExcelRT Plugins	22

Chapter 8

ExcelRT Cloud 8-1

Define Apps	2
Define Users	3
User Experience	3
Custom File Manager	5
Settings	6
Feedback	7
Cloud Sharing	8
User Notify and Alert	9
Batch User	10
Plugin Folder	12
Update an App	14
Custom Login	15
Browser & Device User Experience	16
ExcelRT Control Panel	17
ExcelRT Cloud Pricing	19
Multiple Vendor Accounts	22
Relocate Vendor Account	22
Safe Activation	23
Create User Account	23
Suspend User Account	26
App Family	27
ExcelRT Plugin	28
User Profile	30
Script Commands	31
Backups	32

Chapter 9

Tutorial 9-1

TravelCalc	1
Convert Workbook	11
Controls and Scripts	19

Chapter 1

Setup ExcelRT

This chapter introduces ExcelRT and the ConvertExcelRT tool. It describes the process used to convert an Excel workbook into a platform-neutral file. That file runs within the free ExcelRT runtime application on Mac or Windows. An ExcelRT file can run in a browser on any computer or mobile device with ExcelRT Cloud.

This chapter describes the process of installing and uninstalling the software on macOS and Windows computers. It shows how a developer can activate ExcelRT Builder with a Serial Number. ExcelRT Builder is used to author, optimize and encrypt a workbook for distribution to customers.



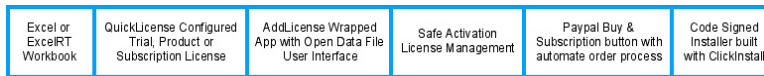
For an end user, ExcelRT is conceptually similar to a run-time version of Microsoft Excel without authoring features. ExcelRT does not support every feature of Excel or other spreadsheet applications, but most commonly used features are available. Some Excel workbooks can be converted to ExcelRT with modest effort.

An ExcelRT file can be authored in Microsoft Excel, then converted to ExcelRT with a conversion tool or authored directly in ExcelRT Builder.

Introduction

Microsoft Excel is a powerful spreadsheet-authoring tool. With years of continuous development and millions of users worldwide, it is the undisputed leader in spreadsheet software. Excel allows users with limited computer skills and no programming experience to create applications useful to others with a similar business, hobby or interest.

Excel Software makes tools to protect, license and productize Excel workbooks that can be distributed and sold to customers that have Excel installed on their computer. Starting from a finished Excel workbook, there are several parts in the process to complete the user experience as illustrated below.



As you can see, most of the productization process is the same for an Excel or ExcelRT based application. If you have already developed an Excel application it can quickly be transformed to a finished, sellable application. If you later switch to an ExcelRT based application, almost all of the tools and process are the same.

The strength of Excel as an authoring tool often presents a weakness as a distribution platform for the finished workbook. Issues faced by a developer selling an Excel workbook are discussed below. ExcelRT addresses many of these issues to offer a better distribution platform for smaller workbooks.

Menus and Ribbon

For most applications, the Excel menu command and ribbon features are a distraction to the user working with the finished application. It often gives the inexperienced user the ability to break the application so it no longer functions as expected.

Developers use VBA code to hide the menus and ribbon. Excel features can hide formulas and determine editable cells or to validate data entry. The ExcelRT runtime is designed to produce an end-user application without distracting authoring commands or ribbons.

Platform Support

The full Microsoft Excel application is available for Mac and Windows computers. Linux or mobile users are unable to use your product. ExcelRT runs on Mac, Windows or in a browser on any computer or device.

Excel Version and Platform Inconsistencies

Excel has been under development for over 30 years. Your customers may use many versions of Excel with different supported features. Users may have Excel 2007 through 2020 on Windows or Excel 2011 or 2016+ on Mac. A lightweight version of Excel is available in a browser but cannot be used for a protected, licensed App.

An Excel developer must decide whether to limit product features to those supported by the oldest version of Excel or require users to buy a specific version and limit their potential sales. The developer and user must contend with Excel platform and OS differences.

ExcelRT features are standardized across platforms. The ExcelRT runtime can be distributed with your application so you control the user experience.

User Costs

With an Excel workbook product, potential customers must buy Microsoft Excel. Some users may already have it, but it may not be on every computer where they want to use your product.

If your product sells for \$5000, the cost of Excel is insignificant. If your product sells for \$50, the cost of Excel will significantly reduce your potential market. The ExcelRT runtime engine is free to use and distribute.

Vendor Costs

When selling an Excel Workbook product, a vendor has tech support cost to deal with Excel specific issues like Trust Center settings, different Excel versions and hundreds of user modified preferences. The Excel ribbon has features that can confuse the user or disable the workbook.

ExcelRT provides the same user experience across platforms with controlled user customization. An ExcelRT based application should lower vendor support cost.

Protection

Microsoft Excel was designed to produce documents that are inherently unprotected and easily shared between users. Using Excel alone you cannot protect and license your product. Excel Software makes tools to protect and license your workbook, but some developer effort is required to prepare your workbook before wrapping it into a protected EXE (Windows) or APP (Mac OS X).

Although it can be used independently, ExcelRT was designed for protection and licensing using QuickLicense or AppProtect. The ExcelRT runtime environment has no commands to show cell properties, formulas or decompile an encrypted ExcelRT file. An ExcelRT app is secure by design.

User Knowledge

Some potential customers may have never used Excel. If you have been using Excel for years, you have probably forgotten all the questions that a first time user faces.

The Excel menus, ribbons and popup menus will likely add unwanted complexity and useless authoring features to your application. You'll need to educate your users on how to use your application without getting overwhelmed by complexity.

ExcelRT presents a streamlined interface for your App that lowers the learn curve for a typical consumer.

Vendor Branding

Most vendors want to present their product as a professional application with its own icon and custom interface. You may want your application to be completely self-contained so it can be installed and used without requiring an additional setup process on the computer or device.

The ExcelRT runtime can be installed with your application. Your App can have its own icon, licensing process and no dependencies to external software.

ExcelRT Overview

ExcelRT implements a core subset of the runtime features in Microsoft Excel with royalty-free distribution. The same ExcelRT file can produce a Mac or Windows desktop application or run in a browser from any computer or mobile device with ExcelRT Cloud.

ExcelRT presents a tabbed workbook window showing the cell structure of each spreadsheet. Your App presents a simplified menu consisting of just **Open**, **Save**, **Print** and **Quit** commands on the **File** menu, that's it. A user can open, edit data, use your product features and save their work. Data is stored in a platform neutral, encrypted file.

The ExcelRT runtime engine is not an authoring tool. A developer can author a workbook using ExcelRT Builder or use Excel on Mac or Windows and then convert it to ExcelRT. ConvertExcelRT runs on Windows with 32-bit Excel installed.

A developer can enable the ExcelRT Builder ribbon to author or refine a workbook into a polished product. With ExcelRT Builder, Microsoft Excel is not required.

As an authoring tool, Excel attempts to guess a user's intent when they type data into a cell or it is displayed back to the user. This often allows an author to achieve reasonable results without defining specific cell formats, data validation rules or even ensuring their formulas do not generate errors.

Since ExcelRT runtime engine is not an authoring tool, don't expect it to guess anything about the user's intent. If you want a cell to be displayed as a date, a time or a currency value, give it a specific format. If you want the user to be able to enter a value like 1/2/2016 or Jan 2, 2016, then assign a data validation rule to the cell declaring it to be a date value.

To summarize, if you want data in a cell to be entered, stored, used or displayed as anything other than text within ExcelRT, give it a specific display format and data validation rules.

Design vs User Mode

ExcelRT support two file types, .xml and .ert. An XML file stores data in a formatted text file used by a developer. An XML file is generated from an Excel workbook using the ConvertExcelRT tool or from ExcelRT Builder. An ERT file stores data in a compressed, encrypted file intended for distribution to a user.

If you are a programmer, think of the XML file as the source file for your project and the ERT file as a compiled binary file where the user cannot see or modify formulas.

ExcelRT can only edit one file at a time. If a user double-clicks the ExcelRT runtime application, it presents the file selection dialog to choose an XML or ERT file.

ExcelRT is typically used together with either QuickLicense or AppProtect to wrap an ERT file into a licensed application. The AddLicense wrapping tool included with QuickLicense adds the optional Open Data File user interface window. The user selects a file from the Open Data File interface window to open it into ExcelRT.

Based on the type of opened file (.xml or .ert), ExcelRT is in Design or User mode, respectively. In User mode, the **File** menu has **Open** and **Save** commands.

In Design mode, the **File** menu has additional commands to view or change cell data, conditional rules, range names, etc. For example, the developer could change the cell value, formula, display format or color for a selected cell. If ExcelRT has been activated with a Serial Number, the full ExcelRT Builder environment is available.

Limits and Differences

If your workbook is large, has dozens of sheets or uses the most complex features available in the latest version of Excel, ExcelRT is probably not a good solution for your project. While Excel offers thousands of powerful features, most workbooks use a tiny fraction of its capabilities.

Here are some limitations of ExcelRT that may disqualify it for a specific project. Over time, some limitations may be removed as new enhancements are added. ExcelRT also implements Standard, Calculated and Static sheet types to optimize memory, file size and reduce open and save time.

Current ExcelRT limitations:

- Maximum 15 Sheets per Workbook
- Maximum 999 Columns per Sheet
- Maximum 999 Rows per Sheet
- Maximum 500 Tokens per Formula - “=A1+D2” has 3 tokens A1, + and D2
- No VBA or Macros (use Button and Event driven scripts or Python)
- No UserForms (use DataForm script command or scriptable dialogs)
- No ActiveX Controls, AddIns, 3D Maps or Spark lines
- No External Data or Connections (connect with Apps, Internet or ODBC)
- No Zoom, Freeze panels or Splits
- Desktop Apps can support workbooks up to about 200,000 total cells
- Cloud Apps can support workbooks up to about 100,000 total cells

Formulas are a core feature of any spreadsheet. Formulas generally work the same in ExcelRT. Excel implements hundreds of functions that can be used from formulas. Some functions are only supported in newer versions of Excel.

ExcelRT implements the most commonly used functions from Excel using the same parameters. If a function isn’t supported, you’ll likely see a Message dialog pop up when a converted workbook is opened in ExcelRT so you’ll instantly know that it is not currently supported.

Refer to Excel documentation for information about each function and its parameters. If an error occurs in the function parameter list or the computation of a function, ExcelRT returns ???.

Supported Platforms

ExcelRT runs on Windows 7 through 10 or on Mac OS 10.10 or later. ExcelRT on macOS is a Universal app that runs native on Intel 64-bit or ARM 64-bit computers.

ExcelRT Cloud works on almost any modern browser from any computer or device.

Install on Windows

ExcelRT and ConvertExcelRT each have their own setup file that installs the software and related resources just like any other Windows application.

Install ExcelRT

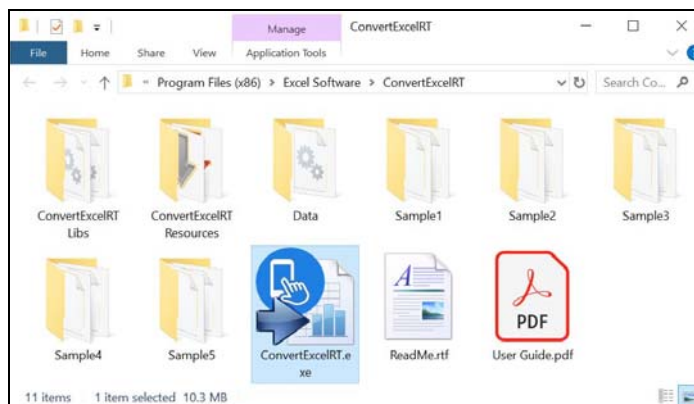
To install ExcelRT on Windows, download and run the Setup file. After installation, use the shortcut desktop icon or the ExcelRT command on the **Start** menu.

Install ConvertExcelRT

To install ConvertExcelRT on Windows, download and run the Setup file. After installation, use the shortcut desktop icon or the ConvertExcelRT command on the **Start** menu. To use ConvertExcelRT you will need 32-bit Excel installed. 64-bit Excel is not directly supported. With newer versions of 64-bit Office, Microsoft appears to be including the 32-bit DLLs so ConvertExcel can be used.

The ConvertExcelRT application, support files and sample projects are installed to this folder. This folder contains the User Guide PDF.

C:\Program Files (x86)\Excel Software\ConvertExcelRT



ConvertExcelRT Folder Installed on Windows 10

Run ExcelRT or ConvertExcelRT

Shortcut icons are installed on your desktop. Double-click an icon to run that application.

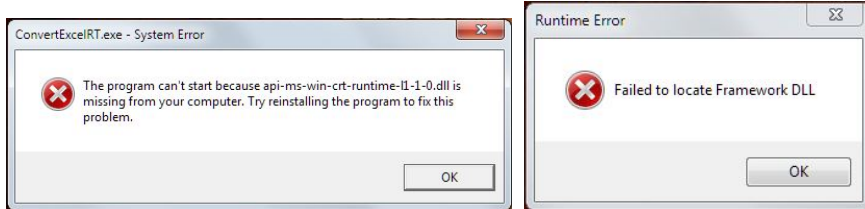
ExcelRT and ConvertExcelRT applications use standard Windows operating system resources. The Windows Update process ensures that your computer has the necessary OS features required by modern software applications.



If you have an older Windows 7 or 8.1 computer and have disabled the normal Window Update process, you may need to do an additional installation step.

Windows DLL Update

If you have an older Windows 7 or 8.1 computer and have disabled the normal Window Update process, you might see an error dialog when attempting to launch ExcelRT or ConvertExcelRT for the first time.



If you see an error dialog like this, close the dialog and see the online instruction on the Missing DLLs page linked from the download page.

Uninstall

To uninstall ExcelRT or ConvertExcelRT on Windows, use the standard approach to remove software from the Windows Control Panel.

Install on Mac

ExcelRT is an application that runs on Mac OS 10.10 or later. The ConvertExcelRT application is not available on Mac since it requires resources that are only available on Windows computers with 32-bit Excel installed.

You can author your workbook using Excel running on a Mac, but the conversion process must occur on a Windows computer. Once you have converted the workbook, it works with ExcelRT running on Mac or Windows.

Install ExcelRT

To install, double-click the ExcelRT setup file. ExcelRT is installed in the Applications folder. You may want to drag the application file to your Dock so it is easy to launch.



Uninstall ExcelRT

To uninstall ExcelRT on Mac, just drag its folder to the trash. You can also drag the ExcelRT.ini file to the trash. It is stored in the Excel Software folder within the Preferences folder in the Library folder of your home folder.

Activation

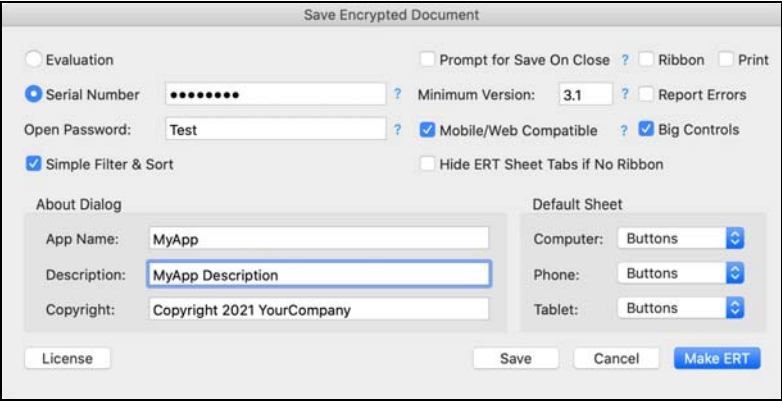
ExcelRT and ConvertExcelRT are free for anyone to download from the www.excelsoftware.com website and use for personal or commercial applications.

ExcelRT can be activated with a purchased Serial Number to give a developer additional rights and capabilities. A developer only needs to activate ExcelRT on one development computer for unlimited distribution rights for all created files.

- Generate Encrypted Files (ERT files)
- ExcelRT Builder
- Distribute the ExcelRT application directly to Customers
- Protect and License an App with QuickLicense or Cloud License
- Free Technical Support by Phone or Email

Workbooks created with an ExcelRT developer subscription can be distributed and used forever, even if the developer subscription is not renewed.

ConvertExcelRT generates an XML file used by developers. When that file is opened into ExcelRT, it can be saved as an encrypted .ert file using the **Save Encrypted** command on the **File** menu. Enter your Serial Number and click **OK**.

A screenshot of the 'Save Encrypted Document' dialog box. The dialog has a title bar 'Save Encrypted Document'. It contains several sections: 'Evaluation' with radio buttons for 'Evaluation' and 'Serial Number' (selected); 'Open Password' with a text field containing 'Test'; 'Simple Filter & Sort' with a checked checkbox; 'Prompt for Save On Close' with an unchecked checkbox; 'Ribbon' with an unchecked checkbox; 'Print' with an unchecked checkbox; 'Minimum Version' with a dropdown set to '3.1'; 'Report Errors' with an unchecked checkbox; 'Mobile/Web Compatible' with a checked checkbox; 'Big Controls' with a checked checkbox; 'Hide ERT Sheet Tabs if No Ribbon' with an unchecked checkbox; 'About Dialog' with fields for 'App Name' (MyApp), 'Description' (MyApp Description), and 'Copyright' (Copyright 2021 YourCompany); 'Default Sheet' with dropdowns for 'Computer', 'Phone', and 'Tablet' (all set to 'Buttons'); and a 'License' button. At the bottom are 'Save', 'Cancel', and 'Make ERT' buttons.

Enter a Serial Number to Save an Encrypted Document

On first use, an Activation dialog is presented to complete the activation process. Enter your Serial Number and Company name, then click **Activate Now**.

A screenshot of the 'Activation' dialog box. The dialog has a title bar 'Activation'. It contains two text input fields: 'Serial Number' and 'Company'. At the bottom are two buttons: 'Activate Later' and 'Activate Now'.

Activate with a Serial Number

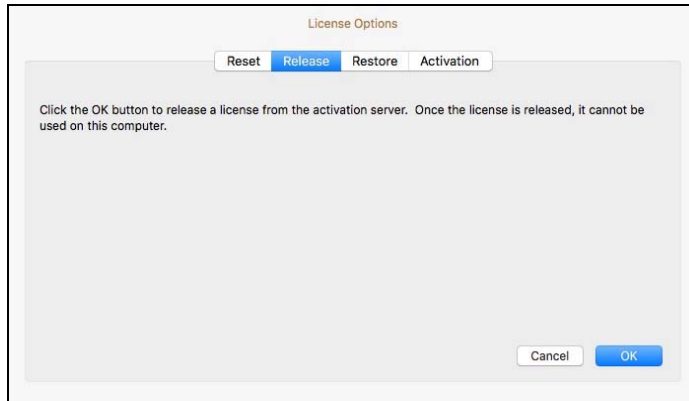
In the unlikely event that your computer is not connected to the Internet, it can still be activated. A Select Activation Type dialog is presented describing how to do a manual activation.

Move a License

ExcelRT is licensed for use on a specific computer during the activation process.

If you purchase a new computer, you can release the license from your old computer so it can be installed and activated on the new computer. From the old computer, launch ExcelRT and present the Save Encrypted Document dialog.

With the **Shift** and **OS** keys pressed, click the **OK** button until you see the License Options dialog. The **OS** key, also called the Command key has an Apple icon for Mac or Window icon for Windows.



Release a License to Use It on Another Computer

Select the Release panel and click **OK** to block the license on the old computer and free it for use on another computer. Now, install ExcelRT onto the new computer.

The License Options dialog can also be presented with the **License** command on the **File** menu.

Restore a License

If ExcelRT is installed, activated and released, the software still resides on the computer but features are blocked since it does not have an active license. To restore an active license, press the **Shift+OS** keys when clicking on the **OK** button in the Save Encrypted Document dialog..

The License Options dialog is presented. Select the Restore panel and click **OK** to restore the active license. The license cannot be restored if it is currently being used on another computer.

Quick Start

Here is an outline of activities to create an ExcelRT workbook. Watch the introduction videos on the ExcelRT home page on the Excel Software website. An extensive video library is available on the ExcelRT panel of the Videos page.

An ExcelRT file can be authored with ExcelRT Builder or converted from an Excel workbook. With either approach, as a developer you will need ExcelRT Builder to refine, test and build an encrypted ExcelRT workbook for distribution to users.

On Windows, install ExcelRT and optionally ConvertExcelRT. If you plan to distribute your product on Mac, also install ExcelRT on a macOS computer.

Install QuickLicense (which includes AddLicense) on a Window and/or Mac OS computer. You can define a license and generate a platform neutral Ticket file from either platform for use on both platforms. Run AddLicense on Windows to output an EXE or AddLicense on macOS to output an APP file.

Alternatively, use AppProtect to build a Mac or Windows application.

To convert an Excel workbook to an ExcelRT file, complete all tutorials from the Tutorials chapter of this User Guide. Alternatively, when authoring a new workbook with ExcelRT Builder skim through the tutorial information then watch a video to get started with ExcelRT Builder.

This User Guide is supplied with ConvertExcelRT. It can be opened from the **Help** menu when the application is running. You can also choose **Sample1**, **Sample2**, from the **Help** menu to open those folders of sample files.

You will likely need to run the conversion process many times. Each time, you will modify your Excel workbook to add or modify cell formats, validation rules and other refinements needed for it to run nicely within ExcelRT.

You can save yourself hours of frustration during the conversion process by reading through the ExcelRT User Guide before you start. It describes differences between Excel and ExcelRT so when a converted workbook doesn't work as expected, you'll know why and how to correct it.

Somewhere during the process, ExcelRT Builder will become the primary development environment for your workbook. Once you start using ExcelRT Builder, remember to keep a backup of the XML source file for your project.

Prepare Excel Workbook for Conversion

Excel may think cells are occupied, even though they appear to be empty.

Before attempting to process a workbook, delete unused columns to the right and unused rows below the used range of cells on each sheet. This may substantially reduce the used cell range of each sheet to reduce conversion time.

From each sheet, drag though about 30 column headers to the right of the last used column. All cells in those columns are now selected. Right-click to present the popup menu and choose the **Delete** command. Now drag through about 30 row headers below the last used row and delete those rows.

This action allows Excel to adjust its range of used cells. Save the workbook.

About This Book

This book covers ExcelRT for Mac, Windows or ExcelRT Cloud. It documents the Windows edition of ConvertExcelRT. Most of the information is the same regardless of the platform. Where information is platform specific such as the installation process, details are provided for each platform.

Most screen shots for this book were taken from ExcelRT running on Mac. The main window and dialogs for the Windows edition are nearly identical.

Support Services

Excel Software provides technical support and encourages user feedback to improve its products and services. Technical and sales questions can be answered by phone, email or through information and demonstration videos on our web site.

Ph. 702-445-7645

Email: info@excelsoftware.com

Web: www.excelsoftware.com

Chapter

2

Develop ExcelRT

With ExcelRT Builder, a developer can author an ExcelRT file directly without using Microsoft Excel or the ConvertExcelRT tool. See the ExcelRT Builder chapter.

Your workbook can also be authored with Excel on Mac or Windows. Regardless of where you create the workbook, you'll need to save it as a .xls or .xlsx file. Do the conversion on Windows with 32-bit Excel installed. The .xlsx file format is recommended since some Excel features are not retained by .xls files.

When ExcelRT was first introduced, all workbooks were authored in Excel and each editing change required use of the conversion tool. This chapter assumes that your project started as an existing Excel workbook. Familiar Excel terminology is used to explain the capabilities of ExcelRT. Today, the conversion tool is seldom used after the initial conversion. Editing and optimization is much faster with ExcelRT Builder.

The conversion process from an Excel file to an ExcelRT file, may take hours or days to complete. Resist the temptation to throw your Excel file at ConvertExcelRT and expect a successful conversion. That approach will likely just waste time and make no progress towards your goal.

Learn about ConvertExcelRT and ExcelRT by running the tutorials in this book and watching the video. Once you have successfully completed the tutorials you will understand the overall process.

Next read this chapter so you'll know how to troubleshoot issues that will occur during your own conversion. You'll also learn how to make dramatic improvements to the performance and usability of the finished product.

Here is the workflow of a typical conversion project.

1. Author workbook with Excel and save as YourApp.xlsx.
2. Drop YourApp.xlsx onto ConvertExcelRT.exe to output YourApp.xml.
3. Open YourApp.xml with ExcelRT Builder to test it.

Before you can follow this workflow, you'll need to prepare your workbook to account for some fundamental differences between Excel and ExcelRT. Excel is optimized to handle very large sheets with many empty cells since this often occurs during the authoring process. The performance of ExcelRT depends on the total number of cells on each sheet, even those that contain no data.

Before attempting to use ConvertExcelRT, determine the minimum columns and rows required on each sheet. Use the Overview feature to restrict the converted cell range to greatly minimize file size and processing time.

Use and test your workbook like a user, then return to Step 1 to improve it. If you examine YourApp.xml with a text editor, you will notice that it contains XML formatted text data. You may not want to distribute that file to a user since all of your formulas and formatting data would be exposed. Before distribution to a user, you'll want to convert the XML file to an encrypted binary file as described in the Deploy ExcelRT chapter.

ConvertExcelRT

ConvertExcelRT is a tool used by a developer to convert an Excel workbook to an ExcelRT file.

It uses a programming interface provided by Microsoft to extract data from an Excel workbook that is saved in .xls or .xlsx format. That programming interface largely determines what data can be extracted and how long the conversion process takes.



The number of used columns times the number of used rows on each sheet will determine the total number of cells. Since a workbook may contain many sheets, the total cell count can grow large.

As the total cell count grows linearly, the ConvertExcelRT processing times grows exponentially. A conversion project usually takes numerous iterations. For each iteration, the developer updates the workbook in Excel and converts it to an XML file that can be opened into ExcelRT.

At some point, the XML file becomes the source document for your project and all editing changes thereafter are done with ExcelRT Builder.

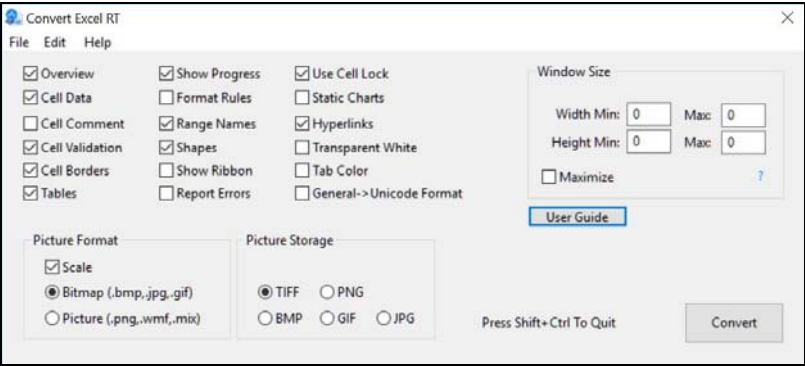
While a fast computer with plenty of RAM can speed the conversion of an Excel file, other strategies are needed to convert a large Excel workbook to ExcelRT.

- **Divide and Conquer** - Use the Overview dialog to process one sheet at a time and work out the issues with that sheet. Reduce processing time by unchecking optional conversion features like Cell Validation, Cell Borders and Format Rules when attempting to isolate and resolve a specific problem.
- **Make it Work, then Make it Pretty** - Show all sheets, column labels and gridlines while you focus on making the workbook functional. There is no need to hide formulas in Excel since there are not visible in an encrypted ExcelRT file.
- **Experiment on Something Small** - Since workbook size dramatically affects processing time, experiment or troubleshoot problems using a small workbook that converts quickly.
- **Use Cell Lock** - Clear this option until issues in the conversion process have been resolved so you can select and view cell properties in ExcelRT.

When launched, ConvertExcelRT presents a window with preferences that determine how a workbook file is processed. Preference data is read from file ConvertExcelRT.ini on launch and saved when quitting the application.

C:\Users\YourAccount\AppData\Roaming\Excel Software\ConvertExcelRT.ini

To process a workbook, click the **Convert** button and select the .xls or .xlsx file. When completed, ConvertExcelRT outputs a file with the same name but .xml extension, then quits itself. Once processing begins you can terminate it by holding down the **Shift** and **Ctrl** keys and waiting for a few seconds.



ConvertExcelRT Main Window

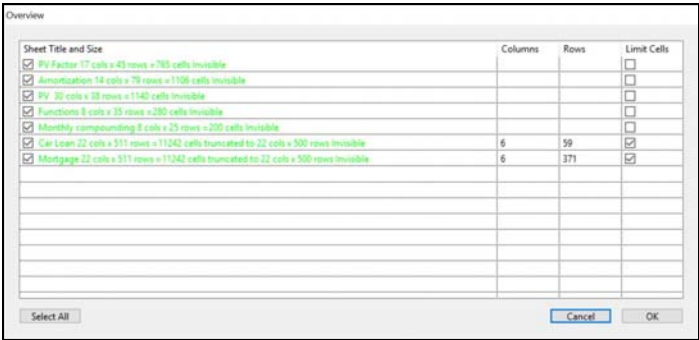
Once you have experimented with and set options the way you want, you can process a file by simply dragging and dropping it onto the ConvertExcelRT shortcut or application icon.

The Window Size options in ConvertExcelRT allow you to control the minimum and maximum window height and width or maximize the window to fit the user’s screen.

Overview Dialog

To prepare your workbook, delete unused columns and rows to minimize the number of cells. The Overview checkbox allows a developer to determine how many cells Excel is using on each sheet or to process a specific sheet to troubleshoot an issue.

Set the Overview checkbox in the ConvertExcelRT window. Now when you process a workbook, an Overview dialog is presented with information about each sheet and the maximum number of columns and rows it requires.



Overview Determines Which Sheets and Cell Ranges are Processed

Sheet names listed in the dialog are colored green, black or red. Red sheets have many cells and will take a long time to process. They'll take a long time to open in ExcelRT and if they are too big they may not open at all, especially if the user has an older computer with limited RAM. Try to reduce the cell count before attempting to convert those sheets.

Use the Overview dialog to process and troubleshoot one sheet at a time in a large workbook. The Limit Cells checkbox can be useful when troubleshooting an issue on a specific sheet. For example, assume your sheet has thousands of cells and takes an hour to process. To experiment with formatting options in cell B2, you could limit processing to 2 columns and 2 rows and reduce processing time to seconds.

Keep in mind that your workbook will not function properly and ExcelRT may even crash when processing just a portion of the sheet especially if a cell references another cell outside of that range.

Use the Columns and Rows fields for each sheet listed in the Overview window to restrict the sheet size. If the bottom right cell used on Sheet 1 is G22, then type G in the Columns field and 22 in the Rows field for that sheet in the list of sheets.

ExcelRT Design Mode

The conversion process creates an XML file that can be opened into ExcelRT in Design mode. The **File** menu contains several commands like **Cell Edit** that may be helpful during the conversion process.

Cell Edit

To present the Cell Edit dialog, select a cell and choose **Cell Edit** from the **File** menu. Early in the conversion process, you may want to clear the Cell Lock checkbox in the ConvertExcelRT tool. Unless you are using ExcelRT Builder, you cannot select a cell to view Cell Data unless that cell is editable.

Design for 100% Zoom

Design your sheets to display nicely at 100% Zoom in Excel. Sheets in ExcelRT are always displayed at 100% scale. Since the same text displayed by Excel and ExcelRT may vary slightly in size, wrapping may occur at a slightly different location. You may need to make cosmetic adjustments in Excel before running ConvertExcelRT.

Pictures

Excel allows a developer to paste images of various formats into sheets and then resize and position them. Different image formats handle transparency in different ways to allow background cell text and colors to show through.

ConvertExcelRT uses a Windows API command to extract those images into an XML file. The API command imposes some restrictions on the quality of the resulting image. ConvertExcelRT has Picture Format, Scale and Picture Storage options that a developer can experiment with to achieve the best conversion results.

For best results, set your sheets at 100% zoom and use an image in the Excel workbook that requires no scaling. Clear the Scale checkbox in ConvertExcelRT since that can degrade the image quality.

If your workbook uses a bitmap type image using BMP, JPG or GIF format, set the Bitmap radio button, otherwise set the Picture radio button in ConvertExcelRT. The Picture Storage option determines the storage size of the picture in the XML document. TIFF is often the best option for good quality at the smallest file size.

If transparency of your image is not retained during the conversion process, then consider setting the transparent areas of your image to pure white before pasting it into the Excel Workbook. Set the Transparent White checkbox in ConvertExcelRT before processing the workbook.

Large or multiple images in a workbook can negatively affect performance in ExcelRT. For some projects it is easier to delete or ignore pictures in the original Excel workbook, then add them after conversion using ExcelRT Builder.

Tables

Excel supports tables to apply special formatting and Sort and Filter features to a range of cells. Some of the style and layout options are intended for a designer that is creating the workbook, but other features like Filter columns are intended for the user of a workbook. There are feature and cosmetic differences between Excel versions and platforms (Mac and Windows).

ExcelRT supports the basic user features of Excel tables. The user cannot alter the style or size of the table, but can enter data, sort and filter it as intended by the designer. In Design mode, you'll notice a **Table** command on the **File** menu allows the designer to view and even change table configuration parameters.

Unlike Excel, ExcelRT cannot infer the type of data in a cell. Unless specified with a Validation rule, it assumes all data is just plain text. If your table contains date or numeric values, Filter options will not work as expected until you assign a Validation rule to each cell so ExcelRT knows what type of data it contains.

Assume you have a table in an Excel workbook that contains a column of dates, but you have not yet assigned any validation rules to those cells. If you type 1/1/2016 into a cell, Excel internally stores the value 42370 to represent that date value. By inferring the data type of that cell, Excel allows you to type a familiar date format and displays the value with a date format, even though it is storing a numeric value in the cell.

Now assume you convert this workbook to an XML file and open it into ExcelRT. Cells with existing date values will display as a number. If you type a new date like 1/2/2016 that text is stored in the cell. If you present the Filter dialog by clicking in the top header column, date comparison features will not work as expected. For tables to work correctly, you must assign a validation rule to each user entered value.

Encrypted File

When launched, ExcelRT starts in Design mode if an XML file is opened. If the user opens an ERT file, it opens in User mode.

In Design mode, choose the **Save Encrypted** command from the **File** menu to output an encrypted file with ERT extension. This command presents a dialog to customize features available to the user.

When ExcelRT is in Design mode with an unprotected .xml file open, a developer can examine the properties of a cell with the **Cell Edit** command. Before outputting a .ert file, use the Build Dependency command if you have made any manual cell editing changes.

From Design mode, the recalculation algorithm can be changed from a command on the **File** menu. For most workbooks, Sheet Recalculation works fine and is much faster.

When your product is finished, tested and ready for distribution you will probably want to add licensing and user interface features to it as described in the Deploy ExcelRT chapter.

Exceptions

During development, you may encounter an exception that causes ConvertExcelRT or ExcelRT to crash. An exception can be triggered by a programming error in one of those applications, using an unsupported Excel feature or reference errors within your workbook.

If your workbook works fine in Excel but crashes ExcelRT, you will likely need to make a design change in your Excel workbook to correct the problem.

For example, ConvertExcelRT attempts to guess how many columns and rows are required on each sheet based on cells that contain data. You'll see the value of those guesses in the Overview windows. A sheet with too many cells may cause an exception during the conversion process or later when opening it into ExcelRT. From the Overview dialog, restrict the number of Columns and Rows on each sheet to the minimum required.

When defining filters or conditional rules you can enter a range beyond the number of columns and rows required by the sheet. ExcelRT does not do graceful range checking so an exception will occur when referencing a cell outside of the defined column and row limits of a sheet. This type of error should be caught and fixed when testing your workbook so a user never sees an exception.

Performance

Excel has been under development for decades and is highly optimized for performance of calculations, file size and read and write time especially on large sparsely populated workbooks. Your first attempt to convert a workbook to ExcelRT may result in a much larger file that opens slowly and takes longer to calculate.

There are likely to be several design improvements you can make to dramatically improve performance. The total cell count and calculation algorithm used will likely have the biggest impact on performance.

Assume you have a sheet with 20 visible columns and 80 rows. Perhaps below those cells you use some hidden rows to do intermediate calculations in an area 80 columns wide by 20 rows. This sheet requires 80 columns by 100 rows or 8,000 cells.

Now assume those same hidden cells were placed in a 20 column wide by 80 row tall area directly to the right of the visible cells. Now the sheet requires 40 columns by 80 or 3200 cells. The file size may be 60% smaller and the calculation performance substantially better.

When encrypting an XML file to ERT format for distribution, it also compresses the file size by as much as 80%. This can dramatically reduce open and save time.

Refer to the Recalculation Algorithm section to learn how to dramatically improve data entry performance.

Feature Differences

ExcelRT generally replicates the runtime features of Excel, but some of the more advanced or specialized features are not supported. Excel often supports multiple ways of doing the same thing, but ExcelRT may only support one approach. A workbook that runs fine in Excel may require modifications for use by ExcelRT.

Cell Format

Excel formats cell data using a format string. If you present the Format Cells dialog for a selected cell, the developer can choose from Categories like Number, Currency, Date, Time, Percentage, Fraction, etc. A developer can even create custom formats.

ExcelRT uses the same format string to format its cells. Most predefined formats provided by Excel should also work in ExcelRT, but some custom formats may not work as expected. Use Currency instead of Accounting formats. In addition to the Text Format string, ExcelRT may require other options like Left or Right justify to get the exact data presentation you want.

Functions

Excel supports a huge list of functions that can be used within formulas. ExcelRT implements the most commonly used functions that have been supported by Excel for many years. Unsupported functions pop up a message dialog in ExcelRT.

When using ExcelRT Builder, the Function Help dialog can be presented from the Toolbar. This dialog shows an alphabetical list of all supported functions.

Range References

Excel supports range references like `A1:C5` that can be used within a parameter of several functions. ExcelRT also supports range references, but only when used as the complete parameter.

For example, Excel allows the formula `=SUM(A1:C5 + D7)`, while ExcelRT requires that the formula be rewritten as `=SUM(A1:C5, D7)`. Notice how Excel allows the range to be used as part of a calculation, while ExcelRT requires that it be the entire parameter.

Range Reference Function

ExcelRT does not support a function used to define a range reference. For example, the range `B2:INDEX(Fruit, 5, 3)` is not supported.

3D References

Excel supports 3D references where you can do something to a cell or even to a range of cells across multiple sheets. For example `Sheet1:Sheet3!A5` references cell A5 on Sheet1, Sheet2 and Sheet3.

ExcelRT also supports 3D references as long as that reference comprises the entire parameter of a function. For example, `=SUM(Sheet1:Sheet3!A5 + D7)` is not supported, however `=SUM(Sheet1:Sheet3!A5,D7)` is supported.

Formula Parameter Nesting

Excel allows an expression `=FLOOR(A4,1)=TODAY()-1`, but ExcelRT requires it to be rewritten as `=FLOOR(A4,1)=(TODAY()-1)`. Notice the extra parens around the part after the equal sign to ensure that part of the expression is evaluated before the logical comparison is made. ConvertExcelRT adds the extra parameters for you to the original formulas, but you'll need to add you own when manually changing a formula within ExcelRT.

Calculation Precedence

Excel gives precedence to `*` and `/` over `+` and `-` in a calculation. ExcelRT processes calculations left to right unless you include parens. Therefore, the calculation `=1+C3*5` from Excel, needs to be converted to `=1+(C3*5)` to give the same result in ExcelRT.

Range Selection

Excel allows an author to select and do operations on an individual cell or a range of cells. ExcelRT allows the user to select or edit one cell value at a time. An ExcelRT user is not changing the workbook, just entering and viewing data.

Data Entry

As an authoring tool, Excel tries to guess the type of data entered by a user (Text, Numeric, Date, Time, etc.) and assigns an appropriate cell format. For example, a time is internally stored as a floating-point number, but displayed using date and time formatting. Sometimes Excel will guess wrong leading to undesired formats or error conditions. ExcelRT doesn't attempt to guess the format so data is displayed exactly as entered unless the developer has assigned a specific format.

Array Formulas

Array formulas were recently added to Excel, but are not supported in ExcelRT.

Data Validation

Using Excel, a developer can assign data validation rules to user entered data. That is especially important for ExcelRT so it can accept user-entered data and store it correctly. For the correct display and entry of a date like 1/25/1960, the cell must have a date format and a validation rule that only accepts a date entry.

Pictures & Form Controls

ExcelRT supports pictures and form controls added to an Excel document. Within ExcelRT runtime environment, these objects that cannot be moved or resized except by scrolling cells left or right or resizing columns or rows.

The Transparent White option in ConvertExcelRT determines whether or not the white color in a picture is transparent allowing the background cell data to show through. If the picture contains some white color that you don't want to be transparent make that color slightly off-white with an image editor before adding it to your Excel document.

Background Images

Excel has the ability to assign a background image to each sheet that shows through from behind the cells. ConvertExcelRT cannot extract the background image from an Excel document.

In some cases a large background image within your Excel document may slow processing within ConvertExcelRT. Once you have created the XML file, use the **Background Image** command from the **File** menu in ExcelRT to add a background image if needed. ExcelRT Builder has a button in the Toolbar to add or remove a background image. Since you'll need to do this after each conversion, ignore the background until all other conversion issues have been addressed.

Structured Table References

Excel supports structured references within cells to refer to a column of cells. For example, `Table1[ColumnName1]` identifies a range of cells in a table named `Table1` for a column with header name `ColumnName1`. If the referencing cell is within the table, you can omit the table name and just use `[ColumnName1]`.

ExcelRT also supports basic structured table column references like this. Excel also supports more complex references like `[@ColName]` or `[#RowName]` or `Table1[[#Total],[SaleAmt]]` but these are not supported by ExcelRT.

Text Across Cells

Excel allows text to be displayed across multiple cells, either by merging those cells or just allowing text from one cell to spill across to other cells. ExcelRT supports merged cells. ExcelRT truncates text if it doesn't fit within its assigned cell. From Excel, select the range of cells required to display the text and select the **Merge Cells** command from the ribbon.

Stylized Text

Excel allows different words within cell text to have different fonts styles and colors. For example, *The Red Cat* could have the word *Red* in red color with bold italics and the rest in black color with plain text. ExcelRT supports a single font style and color that applies to all text in the cell.

Within Excel, if you apply multiple styles or insert a carriage return within the text of a cell, then the style information is stored in different way within the workbook and cannot be extracted by ConvertExcelRT. The result is black text on a white background where all style information for that cell is lost.

To fix a cell with mixed styles within Excel, copy and paste the text into NotePad. Make sure all carriage returns are removed so the text is on one line. Now copy and paste it back into the workbook cell.

Within Excel it is possible to merge several cells, add text and then use different styles and carriage returns to format that text. This approach won't work with ExcelRT. To accomplish the same thing, put parts of that text into different unmerged cells and then apply a single style to each cell.

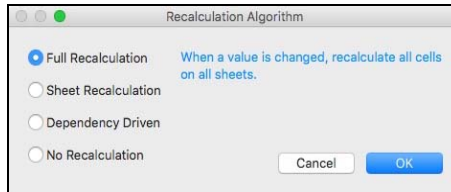
Unicode Text

Cells containing text use the General format in Excel. By default, ExcelRT assumes that a cell with the General format only contains ASCII text. Some human languages like Kanji or special characters require multiple bytes to store each character. Use the Unicode format to read and write cell values that can store non-ASCII text.

During the conversion process, ConvertExcelRT can replace all cell formats of General with Unicode. Alternatively, assign the FormatCells script command to the OnOpen event to assign the Unicode format to a specified range of cells. The Unicode format may affect file size and read and write time for a large workbook.

Recalculation Algorithm

When ConvertExcelRT generates an XML file, it defaults to recalculating all cells on all sheets when any cell value is changed. That approach works fine for small workbooks, but may become unbearably slow on large workbooks for data entry.



Change Recalculation Algorithm Saved with File

The Recalculation Algorithm dialog is presented from the **File** menu in ExcelRT. Sheet recalculation is substantially faster for most multi-sheet workbooks. Each time you generate a new XML file, you'll need to set the Recalculation Algorithm. That selection is stored in an ERT file generated for distribution to a customer.

Understanding and optimizing the recalculation algorithm can have a huge impact on data entry performance for larger workbooks. Recalculation can be controlled by simple script commands. A script command can recalculate a specific sheet or range of cells when the user clicks a button or navigates to a sheet.

For example, assume your workbook has three sheets titled Data, Options and Report. Assume that a large amount of raw data is entered into the Data sheet. Assume the Options sheet allows the user to select various options that instantly summaries and display the results. Finally, assume the Report sheet generates a detailed, printable report of the entered data and options selected.

If you initially use the default Full Recalculation option data entry might be painfully slow since each entered value triggers a complete recalculation of all cells across all sheets. That is completely unnecessary since there are no calculated results displayed on the Data sheet.

Now assume that you default to No Recalculation. On the SheetActivate event for the Options sheet, run a tiny script that sets the recalculation mode to Sheet Recalculation. On the SheetDeactivate event for the Options sheet set the recalculation mode back to No Recalculation. Finally, on the SheetActivate event for the Report sheet, run a script that generates the detailed report.

Your workbook is now instantly responsive to data entry and user selections.

Design for Performance

Microsoft Excel was designed to be an authoring tool for Mac and Windows computers, while the design of ExcelRT focused on its role as a spreadsheet application engine for an application.

By design, an Excel file is very open to user examination and changes to formulas, formatting rules or the addition of sheets, columns or rows. Spreadsheets with many rows and columns, but many empty cells have little affect on workbook size or performance. With ExcelRT, the total cell count has a major effect on performance regardless of whether cells are empty or contain data.

When an ExcelRT file is saved as an ERT for distribution to a user, the formulas and formatting rules can no longer be viewed or changed. The user cannot add sheets, columns or rows unless the designer created a script to implement those features. The workbook size is highly compressed and encrypted compared to an XML file.

Here are some design rules for optimizing an ExcelRT application. While these rules apply to desktop apps, but are especially critical for ExcelRT Cloud.

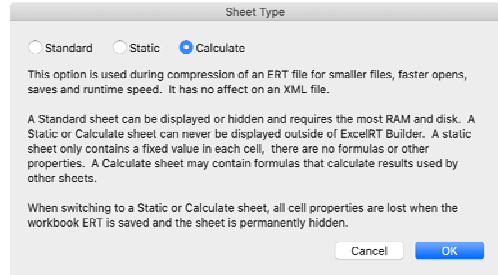
- Minimize the total cell count in the workbook.
- The workbook opens to a Home sheet that should be relatively small.
- Data entry should occur on small sheets to minimize screen redraw time.
- Large sheets should not have editable cells.
- Use recalculation algorithm to improve data entry performance.
- Use a button with script to limit currently visible columns and rows.
- Use Static or Calculate sheets whenever a sheet is always invisible.
- Split one large workbook application into several smaller Apps.
- Share data between Apps through the Plugins folder or via Internet.
- For ExcelRT Cloud apps, minimize the required amount of data entry.

Sheet Type

A workbook can consist of many sheets each containing hundreds or thousands of cells. Each cell contains data about the cell value, formula, dependencies and presentation properties. The quantity of cell data has the biggest impact on the size and performance of an ExcelRT file.

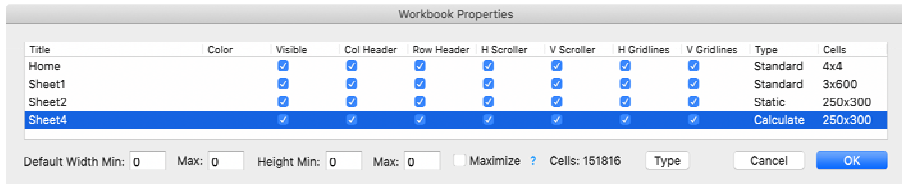
To reduce file size and memory requirements, ExcelRT supports three sheet types:

- Standard
- Calculate
- Static



By default, all sheets are Standard and can be shown or hidden from the user. The cells on a Standard sheet consume the most memory and disk space. If a sheet is never visible to a user, it should be set to Calculate or Static.

Sheet type does not affect an XML file. While in design mode, the developer can view and modify cell data regardless of sheet type. When saved as an ERT file, sheet type determines what cell data is actually stored and later read into memory. Sheet Type is set from the Workbook Properties dialog by selecting the row of data for that sheet and clicking the **Type** button.



A Static sheet only contains the cell value and consumes the least amount of memory or disk space. It does not contain properties to display the cell to a user or formulas used for calculating the cell value. The cell value can be changed from a script command and can be referenced from formulas on other cells.

A Calculate sheet contains cells that do have a formula for calculating the cell value, but consume much less memory and disk space than a Standard sheet.

Fonts

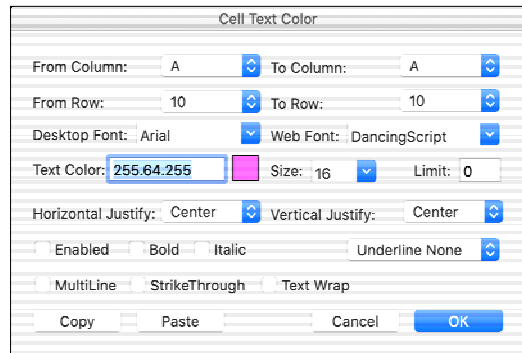
Text fonts are kind of a big subject that you may not have thought about much. When authoring an Excel workbook on your local computer for personal use you may have selected a few fonts that looked good for captions, data entry or notes.

When distributing a document to other users, how text fonts are presented may depend on their environment such as the Mac or Windows OS version they are running, what version of Excel they have and what fonts they have installed.

When distributing an App to a large user base, a developer wants consistency without requiring cosmetic customization by each user.

ExcelRT aims to increase your distribution options to any desktop computer or browser accessing an ExcelRT Cloud application. ExcelRT has built in font handling features that simplify the distribution process.

Each cell in an ExcelRT sheet has both a Desktop and Web font assigned to that cell.



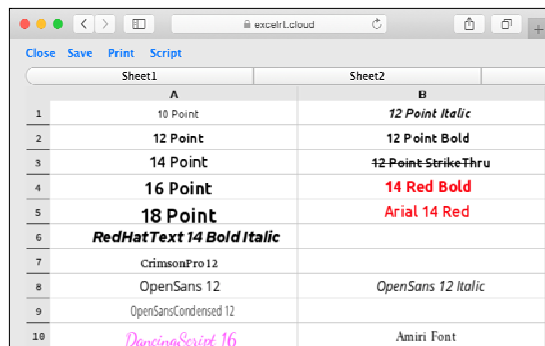
The 'Cell Text Color' dialog box is shown. It has fields for 'From Column' (A), 'To Column' (A), 'From Row' (10), and 'To Row' (10). The 'Desktop Font' is 'Arial' and the 'Web Font' is 'DancingScript'. The 'Text Color' is '255.64.255' (highlighted in blue), 'Size' is '16', and 'Limit' is '0'. There are also fields for 'Horizontal Justify' (Center) and 'Vertical Justify' (Center). At the bottom, there are checkboxes for 'Enabled', 'Bold', 'Italic', 'Underline None', 'MultiLine', 'StrikeThrough', and 'Text Wrap'. There are also 'Copy', 'Paste', 'Cancel', and 'OK' buttons.

Most desktop computers come with standard fonts installed by the OS. The Desktop Font determines the first choice to use when running an ExcelRT file on a desktop computer. If the assigned font does not exist, the OS substitutes a different font.

For ExcelRT Cloud, the core spreadsheet grid is constructed on the server and sent to the local browser for display. While browsers may run on a variety of desktop or mobile devices, the core spreadsheet looks the same.

Server fonts are different than Desktop computer fonts. Each workbook cell can be assigned a specific Web Font.

When designing an App that runs in ExcelRT Cloud, create a sheet with some font and size variations and then load it into a browser to see the end result as illustrated here.



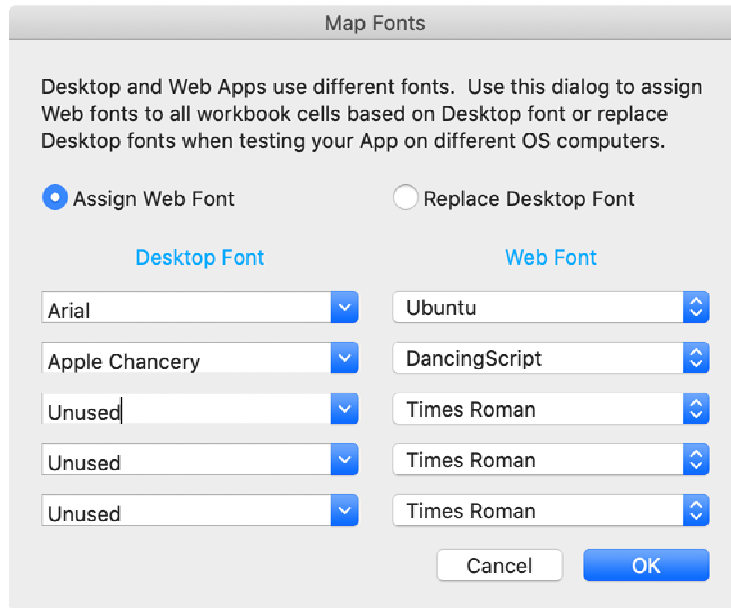
The screenshot shows a web browser window with the URL 'excelrt.cloud'. The interface displays a spreadsheet with two sheets, 'Sheet1' and 'Sheet2'. The 'Sheet2' tab is active, showing a grid with columns A and B. The grid contains various font styles and sizes, including '10 Point', '12 Point', '14 Point', '16 Point', '18 Point', 'RedHatText 14 Bold Italic', 'CrimsonPro 12', 'OpenSans 12', 'OpenSansCondensed 12', 'DancingScript 16', '12 Point Italic', '12 Point Bold', '12 Point StrikeThru', '14 Red Bold', 'Arial 14 Red', and 'Amiri Font'.

	A	B
1	10 Point	12 Point Italic
2	12 Point	12 Point Bold
3	14 Point	12 Point StrikeThru
4	16 Point	14 Red Bold
5	18 Point	Arial 14 Red
6	RedHatText 14 Bold Italic	
7	CrimsonPro 12	
8	OpenSans 12	OpenSans 12 Italic
9	OpenSansCondensed 12	
10	DancingScript 16	Amiri Font

An ExcelRT web application is essentially an ExcelRT file loaded into an ExcelRT Cloud account. Most developers will likely develop and distribute desktop editions of their application first and then later focus on a Cloud edition.

The **Map Fonts** command on the **File** menu can simplify the assignment of Web Fonts to all cells in the workbook based on the assigned Desktop Font. With the Assign Web Font radio selected, select the Desktop fonts used in the workbook on the left side, assigned a comparable Web Font on the right side, then click **OK**.

If the Web Font field of a cell is left empty, ExcelRT Cloud will display that cell value with a default font.



The Map Fonts dialog can also be used to change the Desktop font used by all cells in the workbook by selecting the Replace Desktop Font radio button. This option can be useful when testing your App on different Mac or Windows desktop computers.

Design Mode Interface

The conversion process from an Excel workbook creates an XML file that opens in Design mode within ExcelRT. ExcelRT Builder also stores the project as an XML file. A developer can inspect and modify cell data and workbook options using menu commands or the Builder Ribbon that only exists in Design mode.



While in Design mode, a developer can view the properties of a selected cell. When starting from an Excel workbook, most properties are assigned during the conversion process. Reviewing the cell data can help the designer to understand how ExcelRT works or presents data.

The 'Cell Data: C7' dialog box contains several sections for configuring a cell. It includes fields for 'Comment', 'Value' (containing 'Tom'), 'Formula', 'Dependency', and 'Choices'. Below these are dropdowns for 'Desktop Font' (Calibri) and 'Web Font' (Times Roman), along with 'Size' (13) and 'Limit' (0). There are also color pickers for 'Bkgd Color' (255.255.255) and 'Text Color' (251.25.23), and a dropdown for 'Icon' (None). The 'Format' is set to 'General'. The 'H Justify' is 'Left', 'V Justify' is 'Bottom', and 'Pattern' is 'None'. Checkboxes for 'Enabled', 'Bold', 'Italic', 'Underline None', 'MultiLine', and 'StrikeThrough' are present. There are buttons for 'Borders', 'Validation', 'Bar Properties' (selected), and 'Conditional Bar Properties'. The 'Type' is '0 - Static Text', and 'Text Wrap' and 'Dependency' are checked. A 'Merge Area' field is empty. 'Cancel' and 'OK' buttons are at the bottom right.

Cell Data for Selected Cell

The **Workbook Properties** command on the **File** menu presents a dialog to view or change sheet title, color, visibility and other properties. During development, properties can be changed within ExcelRT Builder. For example, you can show or hide a sheet or click the sheet Title to edit it.

The 'Workbook Properties' dialog box features a table with columns: Title, Color, Visible, Col Header, Row Header, H Scroller, V Scroller, H Gridlines, V Gridlines, Type, and Cells. The rows represent different sheets: Home, Sheet1, Sheet2, and Sheet4. Sheet4 is currently selected and highlighted in blue. Below the table, there are input fields for 'Default Width Min' (0), 'Max' (0), 'Height Min' (0), 'Max' (0), a 'Maximize' checkbox, 'Cells: 151816', a 'Type' dropdown, and 'Cancel' and 'OK' buttons.

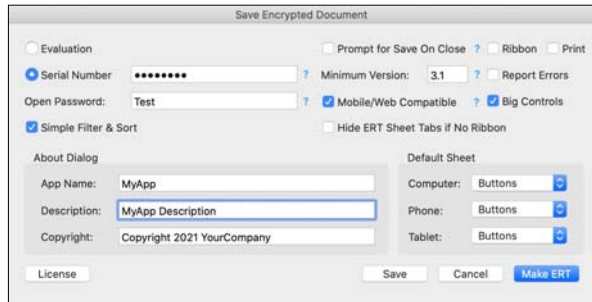
Title	Color	Visible	Col Header	Row Header	H Scroller	V Scroller	H Gridlines	V Gridlines	Type	Cells
Home		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Standard	4x4
Sheet1		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Standard	3x600
Sheet2		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Static	250x300
Sheet4		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Calculate	250x300

Workbook Properties Dialog

User Mode Interface

The user interface in User mode is quite simply. The user can select a cell to edit data, use the **Enter** key to move down the column or **Tab** key to move between columns.

Click on a sheet name across the top to select the active sheet. Excel allows each sheet tab to be colored, while ExcelRT shows that color as a thin bar across the top of the cell grid.



To give an ERT document to the user, set the Prompt for Save on Close checkbox in the Save Encrypted Document dialog. The user is prompted to save when closing ExcelRT. Ignore this checkbox when using QuickLicense or AppProtect. The **Open** command on the **File** menu in the ExcelRT runtime engine is hidden when generating an App with AddLicense or AppProtect.

If your workbook depends on features added in a newer version of ExcelRT, you can set the Minimum Version field before generating the ERT file. Set the Report Errors checkbox to be notified with a dialog if a calculation error occurs.

Most ExcelRT applications provide no editing ribbon to the user. ExcelRT Cloud does not support a Ribbon.

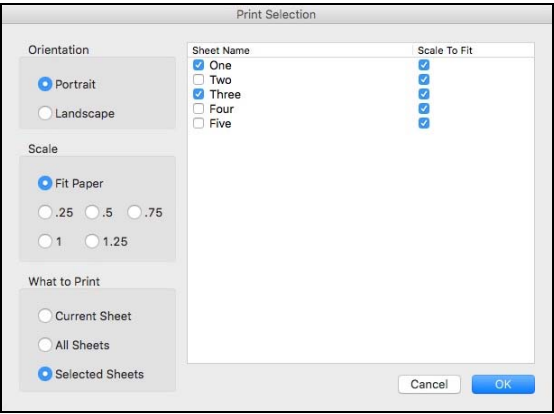


Set the Ribbon checkbox in the Save Encrypted Document dialog to display a tiny ribbon of buttons at the top of the ExcelRT window. The user can select a cell and then click the **Bold**, **Italic** or **Underline** button to apply that style.

Likewise, the cell fill color or font color can be changed to the displayed color in the button icon. To change the default color, double click on the button icon and select from a palette of colors.

The column header, row header, horizontal gridlines, vertical gridlines, horizontal scrollbar and vertical scrollbar can be visible or invisible for each individual sheet in ExcelRT using the Workbook Properties dialog.

The **File** menu includes **Save**, **Print** and **Quit** and commands. Set options on the left side of the dialog from top to bottom to determine the Orientation, default Scale and which sheets to print. If Selected Sheets is chosen, the user can individually select sheets to be printed.



Print One More Sheets

Scripting commands can be used to format and construct a desired image for printing. The **Print** command can be assigned to run a custom script.

Conversion Checklist

Every Excel workbook will require some editing before it is ready for conversion to ExcelRT. This checklist can help to ensure that you've done the minimal amount of preparation required before you begin the conversion process. If you don't understand a specific item below, review this chapter for more details.

1. Using the Overview window in ConvertExcelRT, limit the number of converted Rows and Columns on each sheet to the minimum required.
2. Set each cell format so values align horizontally left, center or right.
3. Replace any custom cell formats with one of Excel's standard formats.
4. Resize columns as needed to fit maximum data width in each cell or alternatively merge cells to prevent truncated displayed data.
5. Add parens to calculations for formula parameter nesting and calculation precedence.
6. Move hidden sheets to the right of all unhidden sheets.
7. Every cell on every sheet should be specifically Locked or not Locked on the Format Cells dialog to determine if it contains a static or editable value. Early in the conversion process, you may want to clear the Use Cell Lock checkbox in the ConvertExcelRT application.
8. Add validation rules to editable cells to constrain user input to safe data ranges and appropriate data types.
9. If recalculations slow data entry, present the Recalculation Algorithm dialog from ExcelRT to adjust when calculations are performed.

Program ExcelRT

Microsoft Excel supports custom programming with VBA. ExcelRT does not support VBA. It does support user or event driven script commands or external commands. A script command also makes it easy to use Python code in a workbook.

With external commands, features can be added to an ExcelRT desktop application with a helper application created with virtually any programming language including Visual Basic, C#, C++, Java, Delphi or Xojo.

A Form Button created on a sheet within the Excel workbook can trigger a button action. User actions like workbook open, save or close can trigger an event action.

Button Actions

Each button can run one or more script commands also referred to as button actions. These commands can read and write to the clipboard or text files, import or export a range of cells as delimited text data or launch an application.

ConvertExcelRT does not generate Button Actions (script commands). To define these commands in ExcelRT, present the Button Actions dialog from the **File** menu while in Design mode. Data from this dialog is stored in a file in the Plugins folder within the folder holding ExcelRT. Since button actions are stored separately, they are retained when doing a new conversion to an XML file with ConvertExcelRT.

When you click **OK** in the Button Actions dialog, the button actions are stored in the ExcelRT file. The commands can be processed when the user clicks a form button on one of the sheets of the workbook.

When you generate an ERT file with the Save Encrypted Document dialog, you do not need to distribute the .actions file in the Plugins folder since button actions are embedded within the file.

For each button that has an action, enter the name of the button, the = character and one or more action commands separated by the | character. Do not include any extra spaces. If the same button name appears on multiple sheets within the workbook, prefix the button name with the sheet name like this `SheetName!ButtonName`.

`ButtonName=Command1|Command2|Command3`

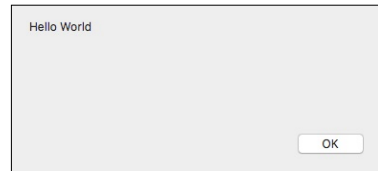
To demonstrate, the `MsgBox()` command presents a message dialog.

`MsgBox(Hello World)`

The button with name `Button 1` presents `Hello World` in a dialog when clicked.

`Button 1=MsgBox(Hello World)`

When the user clicks the button named `Button 1`, the dialog is presented.



Script Commands

Refer to the Script Commands chapter for a description of the most commonly used script commands.

Vendor Commands

ExcelRT Vendor commands are provided by a vendor website for use by customers using their product or family of products. To enable these additional commands, a folder of files provided by Excel Software is licensed and downloaded to the vendor's website. Vendor commands are usually related to Internet, Email or Server features.

Vendor commands can then be used by any application developed by that vendor and work like any other ExcelRT command running on any customer computer or device.

Vendor commands provide convenient email, upload and messaging between ExcelRT solutions. Give your application dropbox-like file storage to communicate messages, images, CSV or even ExcelRT files between computers and people.

Server Setup

The *ExcelRT Vendor Commands* PDF includes instructions to upload a set of files and folders to a Linux or Windows website that supports PHP 5.4 or later. There are some optional PHP and Server parameters that can be adjusted to increase the maximum supported file size which is determined by your website hosting account.

You can optionally enable logging to record a timestamp log of vendor command calls from your ExcelRT applications. You can even send your own log commands from ExcelRT that can be helpful for debugging purposes since the logs can be conveniently displayed in a web browser.

ExcelRT commands in your application communicate with PHP files stored on your website to send email, upload files and support other features. After placing the files on your site, you will have two URLs that get used by the vendor commands.

- ServerURL
- StorageURL

RegisterServer

This command registers a server to implement all other commands in this section. An ExcelRT script will typically call this command once in the OnOpen event. You could use multiple servers by calling this command again when any commands that follow will use that server.

```
RegisterServer(ServerURL)
```

Log

This command logs the supplied Data value to the server if logging has been turned ON. For example, your script could retrieve the user's Serial Number and log it each time an ExcelRT file is opened.

```
Log(Data)
```

Data can be any plain text or the name of a variable that contains text.

Upload

This command is used to upload a file from the Plugins folder on the device running ExcelRT to the `upload` folder on your website. `FileName` is the name of a file in the Plugins folder.

`ResultVar` is the name of a variable into which the return status of the command is stored. If the command is successful, the word `SUCCESS` is returned, otherwise the response string starts with `FAILED`.

```
UploadPluginFile(FileName,ResultVar)
```

This example uploads `Report.jpg` from the device to the `upload` folder on your website.

```
UploadPluginFile(Report.jpg,Result)
```

This command will delete the named file from the `upload` folder on your website. If the file was found, `ResultVar` will contain `SUCCESS`. If the file was not found, then `ResultVar` will contain `FAILED`.

```
UploadPluginFileDelete(FileName,ResultVar)
```

Email

There are three commands for sending email. The `EmailSend` and `EmailSimple` commands use your local web server to send an email. The `EmailServer` command forwards the email message to a remote email server that sends the message.

As useful as email is, it is not always a reliable communication mechanism since there are multiple nodes in the process that can affect the outcome and are beyond your control.

The server hosting your website will determine how quickly the email message gets sent out or if it gets sent at all. Some shared hosting plans will queue outgoing email messages for several minutes or sometimes over an hour before they get sent. The receiving ISP gets to decide whether to receive or reject your email message or to potentially send it to a spam folder.

This command will send an email message to one or more recipients. The message may have one or more attached files from the Plugins folder on the device running ExcelRT. FromEmail is the sender email address. FromName is the sender name and can be an empty parameter.

ToEmail is the receiver email address. To send the same message to multiple email addresses, separate each email address with a semicolon in the parameter as illustrated here: excel@spinn.net;excelsw@cox.net.

Subject is the subject line of the email message. Body is the text within the body of the email message.

The Attachments parameter is zero or more semicolon-separated filenames from the Plugins folder of the sending device.

ResultVar is the name of a variable into which the return status of the command is stored. If the command is successful, the word SUCCESS is returned, otherwise the response string starts with FAILED.

```
EmailSend(FromEmail,FromName,ToEmail,Subject,Body,
Attachments,ResultVar)
```

Here is an example of this command.

```
EmailSend(hwh@spinn.net,Frank,excel@spinn.net,Subject,My
Message,File1.jpg;File2.jpg,Result)
```

Do not use commas within the body of an email message. To include multiple lines in the email message, use a variable named Message in this example that contains text with the # character to represent a carriage return as illustrated here. If you must include a comma in the body of an email message, use ~ instead and it will be replaced by a comma.

```
DefineVar(Message,Hello Tom##Third Line#Fourth Line##Regards#John Doe)
```


The body of the email message will appear like this:

Hello Tom

Third Line
Fourth Line

Regards
John Doe

By default, the `EmailSend` command assumes that you are sending a Plain text message. If you want to use HTML formatting tags, then enclose the Body of the email text with `<html>` and `</html>` tags as illustrated here.

```
EmailSend(hwh@spinn.net,Frank,excel@spinn.net,Subject,<html>Hi Tom<p>How is it  
going?<p>Regards~<br>Excel Software</html>,,Result)
```

The `EmailSimple` command has less features but is likely to be supported by more hosting accounts. It does not have a `FromName` or `Attachments` parameter. The `Body` parameter can only contain plain ASCII text, no HTML formatting is allowed. The `ToEmail` parameter can only have one email address.

```
EmailSimple(FromEmail,ToEmail,Subject,Body,ResultVar)
```

EmailServer

The `EmailServer` command offers the most flexibility for sending email by connecting to the email server of your choice. If your website does not have a working mail server installed, it may be your only choice.

```
EmailServer(Host,User,Pswd,Port,FromEmail,FromName,ToEmail,Subject,Body,Attachme  
nts,Html,ResultVar)
```

This command will send an email message to one or more recipients using a remote email server like Gmail, Yahoo or your company email address.

The message may have one or more attached files from the `Plugins` folder on the device running ExcelRT. `Host`, `User`, `Pswd` and `Port` provide the credentials required to connect to that server. `FromName` is the sender name and can be an empty parameter.

`ToEmail` is the receiver email address. To send the same message to multiple email addresses, separate each email address with a semicolon.

`Subject` is the subject line of the email message and `Body` is the text within message. The `Attachments` parameter is zero or more semicolon-separated filenames from the `Plugins` folder.

Set the Html field to YES when sending HTML formatted text in the Body of the email message.

ResultVar is the name of a variable into which the return status of the command is stored. If the command is successful, the word SUCCESS is returned, otherwise the response string starts with FAILED.

For testing purposes, if you are unable to communicate with the server assign the value YES in the ResultVar variable when calling this command and a stream of communication messages with the server will be presented in a popup window

Here is an example of how to send an email with the EmailServer command using a Gmail account with an App Password.

```
EmailServer(smtp.gmail.com, john@gmail.com, abcd efgh ijkl  
mnop, 587, john@gmail.com, MyCompany, excel@spinn.net, Test, My  
Message, File1.jpg,, Result)
```

Most email servers publish the host name and port number (usually 587) required for an external mail client to send email through the server. The User field is almost always your email address. The Password field can be a simply password used to log into your email account or a special App Password that you create from an Admin screen for that email service.

Some email servers (Yahoo or Gmail) will only allow access using an App Password that you create within your account. To create an App Password within your Gmail account, first enable 2-factor authentication. Now you can create an App Password and use it in the **Pswd** parameter.

Here are some email server examples:

- **Yahoo** - Host is *smtp.mail.yahoo.com* and Pswd is your App Password. Create an App Password in your Yahoo account.
- **Gmail** - Host is *mail.gmail.com* and Pswd is your App Password. You must first enable 2-Factor Authentication. Now create an App Password in your Google account.
- **Cox** - Host is *smtp.cox.net* and Pswd is your login password.
- **Zianet** - Host is *mail.zianet.com* and Pswd is your login password.

Some email servers may reject email messages sent from your specific website even if the server credentials are correct. Use the Debug parameter to help troubleshoot the issue or contact the company that provides that email service.

Storage

This command provides a customer storage folder on your website for a period of time. Your application can use the storage location to transfer files between computers and people with minimal effort by application users. Those files could be images, reports, CSV data files, etc. using file extensions like .jpg and .csv. Once ExcelRT Commands are setup on your server, the system is designed to be self-managed as long as you don't run out of disk space.

```
PluginToStorage(StorageURL,DaysToStore,FileList,Message,DownloadURL,Result)
```

StorageURL is the URL of a storage folder on your website. It can be located anywhere on the same website. That URL is the first part of a web page URL provided to people receiving the shared files.

FileList can be a string or variable containing a string with one or more semicolon separated file names from the Plugins folder. Each file name must be a valid file name on all computer platforms, must include a 3 character file extension like .jpg, .csv, etc and the name must not contain the dash "-" character.

The DaysToStore value indicates how long the files will be stored on your server until they are automatically deleted. Use 1 to 1000 to indicate the number of days or use 0 to store files forever.

Message is plain or HTML formatted text within the body of the web page generated for the receiver of the uploaded files. The DownloadURL is the name of a variable into which a URL is stored that references a generated web page linked to the stored files. Result is the name of a variable into which SUCCESS or FAILED is returned.

In special situations where many files need to be uploaded or collectively the files are too large to upload with one command, the PluginToStorageAdd command allows an additional file to be appended to an existing DownloadURL.

This command must follow the PluginToStorage command. The DownloadURL must reference a download URL that has not expired. The list of file links in the download page is extended each time a new file is added.

```
PluginToStorageAdd(StorageURL,FileName,DownloadURL,Result)
```

This command is used to delete the referenced DownloadURL and all files that it references. Generally you don't need to use this command since it will expire anyway based on the DaysToStore parameter used to create it.

```
PluginToStorageDelete(StorageURL,DownloadURL,Result)
```

Message

These commands are used to send messages between ExcelRT application users. For example, imagine you have created three ExcelRT applications named Manager, Office and Agent. Perhaps roaming project estimators use the Agent app. After collecting data from the customer into the app, an Agent can hit a button to send a message to the Manager to review the quote. Once accepted, the Manager app sends a message to the Agent to issue the customer quote. At that point a message is sent to the Office app to send a PO, purchase supplies, issue a work order, etc.

To send or receive messages, the ExcelRT file must first call the MessageAccept command to register the Organization name and a ToList. The Organization and ToList parameters can be literal strings or the name of a variable that contains the string. In the example mentioned above, you might sell your Manager, Office and Agent apps to IBM, HP, GTE and other companies so their messages need to be handled separately on your website.

All messages are sent from and received by users within the same organization. The ToList parameter contains one or more semicolon separated individual or group names. Create and use any alphanumeric (a..z,A..Z,0..9) names you want. All Group names begin with the word `Group` and can be received by multiple users.

```
MessageAccept(Organization,ToList)
```

In this example, the organization name is IBM. Boss and John are individual user names while GroupAgent might be used to send a message to multiple users that want to accept GroupAgent messages.

```
MessageAccept(IBM,Boss;John;GroupAgent)
```

The MessageAccept command needs to only be called once usually from an OnOpen event. If you close and reopen an ExcelRT file, it remembers your Organization and ToList. When you call the MessageAccept command, the file is saved.

If you want the system to generate a unique user name for you, call this command where UserID is the variable name that will contain the assigned name. The assigned name can be saved in a global variable like \$MyUserName or within a cell in the workbook and then read and used in the future.

```
MessageAssignUser(UserID)
```

The MessageSend command will send a message from the user specified by the From parameter to the user or group specified by the To parameter. The Days parameter should be an integer value that determines when a message is automatically deleted by the system whether it has been received or not. Give it the number 10 to self-delete after 10 days.

The Type and Body parameters are arbitrary text that you provide. This can be anything you want except for the ; character. The From, To, Days, Type and Body parameters may be a literal string or the name of a variable. The MessageID parameter is the name of a variable into which the generated MessageID is stored.

```
MessageSend(From,To,Days,Type,Body,MessageID)
```

For the most part, your application doesn't need to delete messages that it generates since they expire after some time period anyway. A message sent to a specific user is deleted as soon as that user gets the message. Multiple users may accept a group message, so it is not deleted when a specific user gets it.

To delete a message that you have sent, use the MessageID received during the MessageSend command. Call the MessageDelete command with that parameter.

```
MessageDelete(MessageID)
```

The MessageGet command returns the next available message that your application is accepting. All the parameters in this command are variable names. If Result returns SUCCESS, the other parameters provide the message details. If Result returns NONE, no message is available.

```
MessageGet(To,From,Type,DateTime,Body,Result)
```

Most applications will not call the MessageGet command directly. Instead, add an OnMessage event that runs a script to respond to incoming messages. If your script contains an OnMessage event, then once a minute ExcelRT will automatically poll for existing messages and call your script to process it if one is found. Use this command to get the data of the retrieved message. All parameters are variable names and exactly match those in the MessageGet command.

```
MessageReceived(To,From,Type,DateTime,Body,Result)
```

Here is an example script that polls for a message each minute, then pops up the Body of the message in a dialog if one is found.

```
OnMessage=MessageReceived(To,From,Type,DateTime,Body,Result)|MsgBox(Body)
```

By default, your application will automatically poll for messages sent to the ToList in your MessageAccept command if your script contains an OnMessage event. You can disable that polling process by calling the MessageState command with OFF for the State parameter or enable it again with ON for the State parameter.

```
MessageState(State)
```

If your application disables polling and then enables it again later, all messages sent during that time interval are ignored. For example, imagine you go on a two-week vacation. When you return you might want to click a button in your app that calls MessageState(OFF) and then MessageState(ON) to discard all messages sent while you were away.

EXAMPLE

This example script registers your website to handle ExcelRT Vendor commands. It registers your organization and a couple users, then sends a message when you click Button 1 on the workbook sheet. Within a minute, you should see a dialog presented that says "Work Harder".

```
OnOpen=RegisterServer(http://www.yourdomain.com)|MessageAccept(IBM,Boss;GroupWorker)
OnMessage=MessageReceived(To,From,Type,DateTime,Body,Result)|MsgBox(Body)
Button 1=MessageSend(Boss,GroupWorker,2,Urgent,Work Harder,MessageID)
```

In real life the Boss and Workers might be running on different computers or devices located in different cities.

Query

This ExcelRT command is used to query a formatted text file on a website for a matching row of data, then return one or more delimiter separated fields of data from that row.

The Query command requires that your script has already registered your website with the RegisterServer command.

```
RegisterServer(ServerURL)
```

The Query command assumes that query.php and the data file referred to by the DataFile parameter are stored in the excelrt_commands folder on the vendor website.

Query(DataFile,RowDelimiter,ColDelimiter,MatchField,MatchValue,ResultDelimiter,StatusVar,ResultVar,Field1,Field2,...)

- DataFile is the file name containing the data that resides in the same folder as the PHP file.
- RowDelimiter is CR, LF or CRLF.
- ColDelimiter is COMMA, SEMICOLON or TAB.
- MatchField is the name of the field to match.
- MatchValue is the value of the named field to match.
- ResultDelimiter is the delimiter used to separate result values including COMMA or SEMICOLON.
- Status is a variable that stores the status of the command of SUCCESS, FAILED or NOMATCH.
- ResultVar is the variable name where the result is stored.
- FieldX can be one or more named fields separated by commas for which the value is returned in the ResultVar parameter.

Example: Test.txt

ZipCode	Material	Labor	Equipment
01001	0	21	0
01002	5	15	2
01003	4	15	1

Assume this file uses CRLF to separate rows, a Tab to separate columns and you want the Labor and Equipment values separated by a semicolon for the ZipCode 01002.

Use this command:

Query(Test.txt,CRLF,TAB,ZipCode,01002,SEMICOLON,StatusVar,ResultVar,Labor,Equipment)

The following value is returned in ResultVar variable and SUCCESS in StatusVar.

15;2

PythonServer

This ExcelRT command runs a Python script on a server from a desktop application or ExcelRT Cloud. The user does not need to install Python.

```
PythonServer(Script,Input,Output,ScriptDataVar*,InputDataVar*)
```

Run a Python script on a server giving it an optional Input parameter and storing the results in the Output parameter. Script refers to an existing script on the server to run. Input is a short input parameter sent to that script.

ScriptDataVar is an optional variable containing Python script commands to be sent to the server that are temporarily stored in a file and run. InputDataVar is an optional variable containing data sent to server and temporarily stored in file then read from the script.

As with all ExcelRT Vendor commands, your script must first register the server.

Cloud Sharing

A **Cloud** button on the Open Data File window of an ExcelRT based application can access a Cloud Sharing account. That feature is used to share or backup ExcelRT workbook files between computers and users.

This section describes how to programmatically access almost any type of file in a Cloud Sharing account from within an ExcelRT workbook using script commands.

To access a Cloud Sharing account, you will need the Cloud ID and either a Read or Write password. To upload or delete files, you will need the Write password. This command returns the file count, allowed number of files, allowed file size, list of file names, file access and status into named variables.

```
CloudSharing(CloudID,CloudPswd,FileCountVar,FileAllowedVar,FileSizeVar,FileListVar,AccessVar,StatusVar)
```

The FileList is a | separated string of file names. If StatusVar contains SUCCESS, then AccessVar contains the value READ, WRITE or SUSPEND. If an error occurs, StatusVar contains FAILED, NOVENDOR, NOACCOUNT or BADPASSWORD.

This command deletes a named file from a Cloud Sharing account provided the correct Cloud ID and Write password is supplied. StatusVar returns SUCCESS, FAILED, NOVENDOR, NOACCOUNT or BADPASSWORD.

```
CloudDelete(FileName,CloudID,CloudPswd,StatusVar)
```


This command downloads a file from a Cloud Sharing account into the Plugin folder. StatusVar returns SUCCESS, FAILED, NOVENDOR, NOACCOUNT or BADPASSWORD.

```
PluginFileFromCloud(FileName,CloudID,CloudPswd,StatusVar)
```

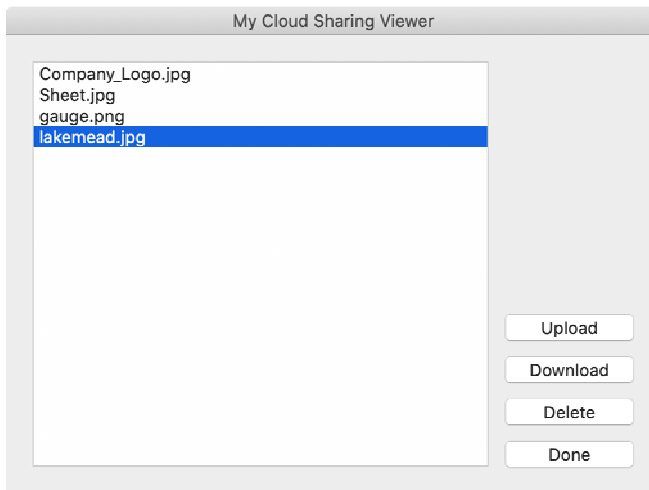
This command uploads a file to a Cloud Sharing account from the Plugin folder. StatusVar returns SUCCESS, FAILED, NOVENDOR, NOACCOUNT or BADPASSWORD.

```
PluginFileToCloud(FileName,CloudID,CloudPswd,StatusVar)
```

This command presents a window to view, upload, download and delete files of specified type in a Cloud Sharing account.

```
CloudViewer(CloudID,CloudPswd,Title,FileTypes,Btns)
```

CloudID identifies the Cloud Sharing account. CloudPswd can be the Read or Write password for that account. If the Read password is provided, the **Upload** and **Delete** buttons are disabled.



The Title field names the dialog that lists files from the Cloud Sharing account. FileTypes is a list of one or more semicolon separated file extensions of the form. Only those file types are display or can be uploaded by clicking the **Upload** button and selecting a file from disk.

```
.jpg;.png
```

Refer to the Cloud Sharing User Guide for a complete list of supported file types.

Settings Dialog

An ExcelRT file usually runs within a product that includes licensing features and the Open Data File interface window. That window has buttons to manage ExcelRT files.

The **Settings** button presents a developer-configured dialog to collect data from the user.

Field	Value
First Name	
Last Name	
Age	
Sex	<input checked="" type="radio"/> Male <input type="radio"/> Female
Color	Red
Combo	
Section1	
Field1	
Field2	
Field3	

The Settings dialog is configured from AddLicense. The Settings dialog allows the App user to enter or change data from one location, then use that data from all workbook files created with that App. Refer to the QuickLicense User Guide for detailed information.

Setting data is easily accessible to an ExcelRT file using one of these two commands.

```
SettingsLoad(FieldName1,CellRef1,FieldName2,CellRef2,...)
SettingsRead(FieldName1,VarName1,FieldName2,VarName2,...)
```

The SettingsLoad command loads one or more workbook cells with user-entered data from the Settings dialog based on developer-configured field names. The SettingsRead command reads the field values into a named variable instead of a cell.

Event Actions

Event actions can also be defined in the Button Actions dialog. Event actions are triggered when a document is opened, saved, closed or at specific time intervals.

The available event action names are:

- OnOpen
- OnSave
- OnClose
- OnSheetActivate
- OnSheetDeactivate
- OnSecond
- OnTenSecond
- OnMinute
- OnTenMinute
- OnResized
- OnMessage
- OnSheetReplace[x]

This event action saves the document every ten minutes.

```
OnTenMinute=Save()
```

This event action presents a dialog showing 1 to 15 based on the active sheet.

```
OnSheetActivate=SymbolValue(S,SheetID)|MsgBox(S)
```

This event can be triggered when a specific sheet is activated as illustrated here for sheet ID 3.

```
OnSheetActivate[3]=MsgBox(Sheet 3 Activated)
```

Likewise, the event can be triggered when a specific sheet is deactivated as illustrated here.

```
OnSheetDeactivate[3]=MsgBox(Sheet 3 Deactivated)
```

This event action overrides the default Print dialog. If your script includes an OnPrint event, then when the user chooses the Print command on a desktop computer your script is called instead.

```
OnPrint=Sub$MyPrint
```

When the user clicks, double-click or modifies a cell value, a script can run. Notice these events include the SheetID, Column and Row number in the event name. None of these can be variables.

- OnCellClick[SheetID,Column,Row]
- OnCellDoubleClick[SheetID,Column,Row]
- OnCellChanged[SheetID,Column,Row]

This script says Hello when user clicks on cell B4 on first sheet.

```
OnCellClick[1,2,4]=Speak(Hello)
```

This event polls active HTML controls each second for an array of data to be copied. If an ExcelRTcopy element is found in the active HTML content that provides array data, this event is triggered. The event supports multiple active HTML controls per sheet and multiple array names per HTML control.

```
OnHtmlViewer[ArrayName]
```

This event is triggered when a named picture control is clicked. The event can run script command that retrieve the click position within the picture using the PictureClickX and PictureClickY symbols.

```
OnPictureClick(ControlName)
```

Regardless of where a picture is positioned within a sheet or the settings of the horizontal or vertical scrollbars, the top left corner of the picture represents X=1 and Y = 1, while the bottom right corner represents X=Width and Y = Height where Width and Height are the pixel size of the picture control.

External Commands

An external application can send commands to ExcelRT using the clipboard or a command file. The external application has full access to the Scripting engine built into an ExcelRT workbook. To enable this feature, set the Enable External Commands checkbox in the Button Actions dialog, then click **OK**.

After converting a document with ConvertExcelRT, this feature is not set. You need to enable it before distributing the document. When disabled, commands are not processed. When enabled, ExcelRT will poll once a second for commands to process.

Clipboard Command

Each Clipboard command consists of a request string of the form `RequestExcelRT:Command` and a response string of the form `ResponseExcelRT:Data`.

For a clipboard command, the external application writes the request string to the clipboard, waits one seconds then polls the clipboard until it finds the response string.

ExcelRT normally writes `ResponseExcelRT:Done` to the clipboard when finished processing command, unless one of the processed commands uses the clipboard for its output.

File Command

For a File command, an application writes a command to a plain text file named `ExcelRT.Request` in the shared Ticket folder and reads the response from file `ExcelRT.Response`. After processing a command, ExcelRT deletes file `ExcelRT.Response` and writes `Done` to file `ExcelRT.Response`. After an external application sees the response file, it should delete file `ExcelRT.Response`.

The same set of commands used by button actions are also supported as external commands. The only difference is a clipboard command that writes to the clipboard does not get a `ResponseExcelRT.Done` string in the clipboard since the clipboard contains the data the application is requesting.

To demonstrate, enable External Commands in ExcelRT from the Button Actions dialog and copy the line of text below into the clipboard.

```
RequestExcelRT:ExportDataToClipboard(A2:B3,Comma,LF,NQ)
```

The command has been processed. Now paste the clipboard into a text editor and you'll see data something like this coming from cells A2 to B3 in the active sheet of ExcelRT:

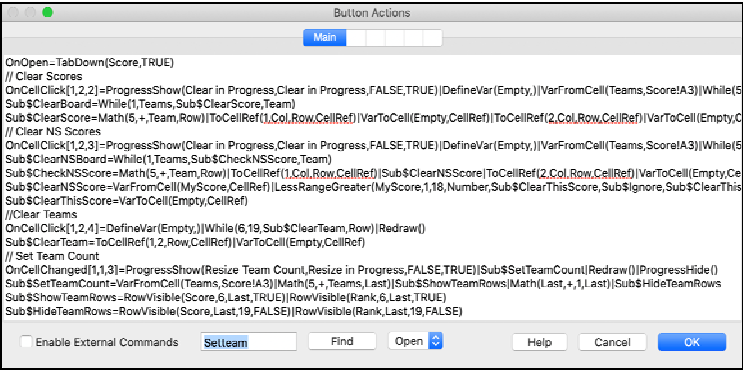
13,91
25,19

Be careful on Mac when using using TextEdit to test clipboard or file commands that include double quotes. By default, TextEdit replaces them with smart quotes. These are not the same as double quotes. Turn off Smart quotes by clearing the checkbox in the Preferences dialog for TextEdit.

Create a Script

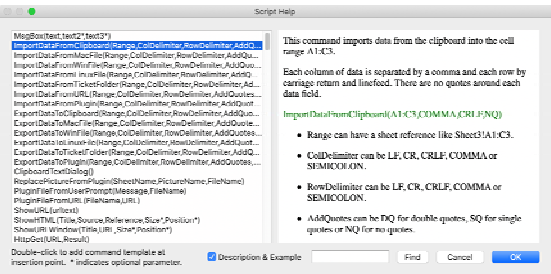
The **Button Actions** command on the **File** menu presents the Button Actions dialog. This dialog shows all the scripts used by the workbook.

A script is a string of text commands that run when a button is clicked or some type of event occurs. Each command consists of the command name followed by open and close parens that may contain one or more comma separated parameters without any extra spaces.



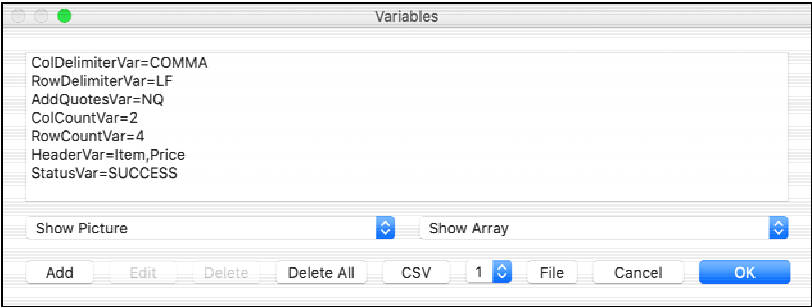
To locate a command in your Script starting from the cursor location, type a string into the edit field at the bottom left and click the **Find** button.

The **Help** button presents a dialog to locate, review or insert a command template at the cursor location.



Variables

The **Variables** command on the **File** menu presents the Variables dialog. The dialog shows the name and value of each variable currently defined by scripts that have run.



Variable Names and Values

While your script is running, local variables are stored in memory. If your document is saved, variables can optionally be stored in the document and later recovered when you open the document. If you no longer need defined variables, call the ClearVars() command in a script.

Pictures and array data are never saved with your document. The **Show Pictures** popup is only visible if a script command has loaded a picture. It allows the developer to select and display a picture. The **Show Array** popup is only visible if a script has defined one or more arrays. It allows the developer to select an array and display the value of its elements.

The **CSV** button presents the CSV Viewer dialog to display the contents of the CSV memory structure used by scripting commands. Use the popup next to the CSV button to display a specific CSV structure.

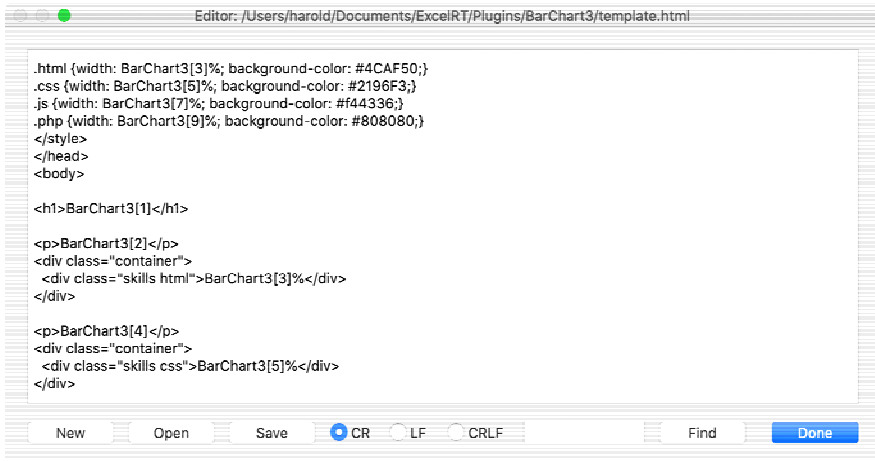
The CSV Viewer dialog box displays a table with 5 columns: 3, 4, 5, and two unlabeled columns. The table contains 24 rows of data. The first row has headers: Product Code, Description, and Sell Price. The subsequent rows contain various product codes, descriptions, and prices. A 'Headers' checkbox is checked on the right side of the dialog. A 'Close' button is at the bottom right.

3	4	5		
1	Product Code	Description	Sell Price	
2	002082175	BILLING CREDIT	0	
3	0189510	"1-1/8"" HDG FLAT WASHER"	0.7	
4	1-778621APE	"1-7/8"" EXLOS" KINGSRAN SEAM TAPE"	15.85	
5	1010000	10X100 TYVER METROWRAP HOUSE WRAP	107.51	
6	101001METRO	10X100 TYVER METROWRAP COMMERCIAL GRADE	188	
7	10101005	10X10 10" P. TREATED #2	0	
8	10110094	MSA FORESTRY KIT	7.8	
9	1012500	10X125 TYVER **COMMERCIAL** WRAP	250.24	
10	1012500-0	10X125 TYVER COMMERCIAL WRAP D	0	
11	1012500	10X125 TYVER DRAIN WRAP	0	
12	1012500-M	BARRICADE HOUSE WRAP 10" X 150"	0	
13	1015000	10X150 TYVER H2AC WRAP	23.51	
14	101500MG	10X150 PACTIV RAINDROF HOUSE WRAP	99.23	
15	10221	"9-1/2"" X 22" LPI20 I-JOIST"	36.28	
16	101	2X4 #2 PINE S4S LF	0.3	
17	10105	2X4 #2 P. TREATED (LF)	0.69	
18	1013	1X4 #3 PINE	0.26	
19	106	2X6 #2 PINE S4S LF	0.38	
20	10605	2X6 #2 P. TREATED (LF)	1.23	
21	1063	1X6 #3 PINE	0.69	
22	1069	1X6 #1 FRT TREATED (LF)	1.23	
23	108	2X8 #2 PINE S4S LF	0.65	
24	10805	1X8 #2 TREATED (LF)	0	

Text Editor

The **Text** button in the Variables window presents a plain text editor that is typically used to create, view or edit text files in the Plugins folder. For example, this can be used to view the contents of a CSV file or edit an HTML files used to build an ExcelRT Plugin.

This text editor can be presented from a script on desktop computers using the PluginTextEditor command. This editor can also be presented within ExcelRT Builder by holding down the **Shift** key while clicking on the Script Edit tool.



Script Line Continuation

A script often has multiple commands, each separated with a | character.

```
Button 1=DefineVar(Msg,Hello)|MsgBox(Msg)
```

The editor limits the length of a line, but you can continue the script on the next line with the continuation characters ... as shown here. Be careful not to add any character such as a space after the ... characters.

```
Button 1=DefineVar(Msg,Hello)|MsgBox(Msg)|...
MsgBox(Another Message)
```


Test a Script

ExcelRT makes it easy to test a script in design mode using the **Run Script** command on the **File** menu. This eliminates the need to add a button on the workbook to test your script.

To demonstrate, present the Button Actions dialog and enter this script:

```
MyButton=MsgBox(Hello)
```

Present the Run Script dialog and enter the button name you used, in this case MyButton. Click **OK** to run the script.



Run a Script

Custom Commands

A custom command can be created with calling parameters. The custom command can then be used throughout the script just like any standard command.

```
Define=CommandName(V1=In,V2=InVar,V3=OutVar,V4=InOutVar,V5=Var)Script
```

A custom command is given a unique name and can have any number of comma separated parameters enclosed with parenthesis followed by one or more scripting commands separated by |.

For example, the first line below defines a command and the second line calls that command when a button named TestBtn is clicked. Notice how the First and Last parameters are literal input value and the Full parameter is the name of an output variable. Also notice how the name of the calling variable FirstLast can be different then the name defined in the parameter list or used within the script that implements that command.

```
Define=FullName(First=In,Last=In,Full=InOutVar)Concat(Full,First,Last)  
TestBtn=FullName(John,Doe,FirstLast)|MsgBox(FirstLast)
```

Each parameter is assigned a type that tells ExcelRT whether to treat it as a literal, a variable and whether parameters are local in scope to that command only.

When defining a command, each parameter is assigned one of these types:

- **In** – Literal stored in a local variable
- **InVar** – Literal or named variable stored in a local variable
- **OutVar** – Local variable initialized to empty and returned in a named variable
- **InOutVar** – Literal or named variable that is copied into a local variable and then returned in a named variable
- **Var** – Normal variable that is never copied into a local variable

Local variables created from command parameters are given a temporary, but unique name while the command is running and purged from the variable list when the command is completed. A developer does not need to think about this when calling a command, but it can be helpful to understand the process when implementing or debugging a custom command.

To understand local variables, add a `Debug()` command to the command script:

```
Define=FullName(First=In,Last=In,Full=InOutVar)Concat(Full,First,Last)|Debug()
```

Notice the variable parameter named First is changed to a name of the form FullName_1_First that is displayed in the presented Variables window. This temporary variable renaming also occurs within the command script itself. If you could view that script while it is running, it would actually look like this:

```
Concat(FullName_1_Full,FullName_1_First,FullName_1_Last)|Debug()
```

Once the command completes, variables FullName_1_Full, FullName_1_First, FullName_1_Last no longer exist.

ExcelRT uses a specific naming convention for local variables. When naming variables, a developer should never use this convention.

```
CommandName + " _ " + Integer + " _ " + ParameterName
```

The benefit of local variables within a command implementation is that these variable names are unique and will not conflict with other variables in the calling script.

There are some side effects of using local variables. If the command script calls a Sub, local variables are not passed to that Sub. When command MyMsgBox(Hello) is called, an empty dialog is presented.

```
Define=MyMsgBox(Msg=In)Sub$MyMsg  
Sub$MyMsg=MsgBox(Msg)
```

Assume that the Msg parameter is changed to type Var.

```
Define=MyMsgBox(Msg=Var)Sub$MyMsg  
Sub$MyMsg=MsgBox(Msg)
```

When MyMsgBox is called as illustrated below, the dialog now says “Hello”.

```
TestBtn=DefineVar(Msg,Hello)|MyMsgBox(Msg)
```

Here is another solution, where one command calls another command and passes in the data. Now the command MyMsgBox(Hello) works.

```
Define=MyMsgBox(Msg=In)MyCmd(Msg)  
Define=MyCmd(Msg=In)MsgBox(Msg)
```

Using a Command as a Parameter

ExcelRT supports many conditional or looping commands that call other Subs.

```
IfDoEvent, IfEvent, While, WhileFields, WhileArray,  
SelectEvent, LessEqualGreater, LessRangeGreater
```

Here is a typical example.

```
TestBtn=IfDoEvent(Var1,=,5,Sub$IsFive,Sub$NotFive)  
Sub$IsFive=MsgBox("Is Five")  
Sub$NotFive=MsgBox("Not Five")
```

The name of the Sub referenced from a conditional or looping command can be replaced with a command as illustrated here.

```
TestBtn=IfDoEvent(Var1,=,5,MsgBox("Is Five"),MsgBox("Not Five"))
```

This example presents a message dialog if Var1 contains TRUE.

```
TestBtn=IfEvent(Var1,MsgBox(TRUE))
```

This example presents three dialogs showing "1", "2" and "3".

```
TestBtn=While(1,3,MsgBox(ID),ID)
```

This example presents three dialogs showing "A", "B" and "C".

```
TestBtn=WhileFields(A#B#C,#,Field,MsgBox(Field))
```

This example presents three dialogs showing "A", "B" and "C" where Item is a variable holding an element of an array.

```
TestBtn=ArrayFromList(Array1,A?B?C,?)|WhileArray(Array1,ID,Item,MsgBox(Item))
```

Importing a Script

An ExcelRT file contains a main script that can import any number of other script files. The main script is shown in the Main panel of the Button Actions dialog.

This command allows an external Script file to be imported into the Main script file.

```
ImportScript(ScriptName)
```

Notice how an ImportScript command reads in script from a file named Script1.script that is stored in the Plugins folder.



Additional Script files can be created or opened in the Button Actions dialog by clicking on the Open popup menu. Script files are opened readonly by default, but you can set the Edit checkbox to make them editable.



Notice how Script1.script defines a custom command named SayMyName that is called from the Main script.

When the OK button is clicked in the Button Actions dialog, the Main script is saved to the Plugins folder and all other script files with the Edit checkbox set are also saved. The script from the main and all imported scripts is stored within the ExcelRT file. When you create an ERT, all script code is stored, compressed and encrypted within the ERT file.

Custom Functions

Cells in ExcelRT support formulas with hundreds of standard functions similar to those found in Microsoft Excel. A custom function can be created with script commands and then used in formulas just like a standard function.

Custom functions are generally not called from other scripts unless they are called using the Calculate command. Creating a custom formula is an advanced topic best left to an experienced ExcelRT developer. A mistake in the function script may cause formula calculations to fail, ExcelRT to hang or crash.

When implementing a custom function, never use a script command that presents a user interface control since that blocks the calculation engine from completing. When implementing a function, avoid using any script command that might introduce a delay such as reading data from the Internet.

```
Function=FunctionName(Parm1,Parm2,...)Script
```

The named function is implemented with one or more script commands including an assignment of the result to the function name. Assume cell A1 contains the value 5 and cell A2 contains the formula =AddOne(A1). The calculated result of 6 is stored in cell A2.

```
Function AddOne(Value)Math(Value,+,1,AddOne)
```

Here is how this custom function works. The value in cell A1 is passed into the function in a local variable named Value. Using the Math command, that value is added to 1 and the result is stored in AddOne. Notice that every function automatically has a result variable that is the same as the function name.

This example creates the custom function named Discount that has two parameters, Quantity and Price. The calculated discount is 10% of the total for orders of quantity 10 or more.

```
Function=Discount(Quantity,Price)Math(Quantity,*,Price,Discount)|Math(Discount,*,0.1,Discount)|IfSetVar(Quantity,<,10,Discount,Number,0.00)
```

All function parameters are treated as local variables similar to how local variables are handled within the implementation of a custom command. While the function is running, the local variables are visible from the Variables windows, but discarded when the function completes.

Function parameters can accept a literal value or a cell reference just like other standard functions in a cell formula.

A custom command can use the Python or PythonServer command. That means the full Python programming language can now be used to add new functions to workbook formulas.

Chapter

4

Script Commands

ExcelRT support hundreds of commands to interact with the user, workbook data, the Internet and physical devices like printers. Most commands are available to all ExcelRT workbooks. Some commands like those described in the Vendor Commands section of the Program ExcelRT chapter require additional software to be installed on the Vendor website.

Excel Software develops custom ExcelRT commands for specific industries or applications. Contact Excel Software for information.

Cell Data Import and Export

Here are some of the supported button commands to import and export cell data. Do not add white space like tabs or spaces before or after any parameter.

`ImportDataFromClipboard(Range,ColDelimiter,RowDelimiter,AddQuotes)`

`ImportDataFromMacFile(Range,ColDelimiter,RowDelimiter,AddQuotes,Path)`

`ImportDataFromWinFile(Range,ColDelimiter,RowDelimiter,AddQuotes,Path)`

`ImportDataFromLinuxFile(Range,ColDelimiter,RowDelimiter,AddQuotes,Path)`

`ImportDataFromTicketFolder(Range,ColDelimiter,RowDelimiter,AddQuotes,FileName)`

`ImportDataFromURL(Range,ColDelimiter,RowDelimiter,AddQuotes,URL)`

`ImportDataFromPlugin(Range,ColDelimiter,RowDelimiter,AddQuotes,FileName)`

`ExportDataToClipboard(Range,ColDelimiter,RowDelimiter,AddQuotes)`

`ExportDataToMacFile(Range,ColDelimiter,RowDelimiter,AddQuotes,Path)`

`ExportDataToWinFile(Range,ColDelimiter,RowDelimiter,AddQuotes,Path)`

`ExportDataToLinuxFile(Range,ColDelimiter,RowDelimiter,AddQuotes,Path)`

`ExportDataToTicketFolder(Range,ColDelimiter,RowDelimiter,AddQuotes,FileName)`

`ExportDataToPlugin(Range,ColDelimiter,RowDelimiter,AddQuotes,FileName)`

Here are the valid parameter values. When supplying your own special characters for a column or row delimiter, be careful not to use characters like (, ,) or | which might otherwise conflict with the syntax used by commands. For example, you could use # or & but not |.

- Range is a cell range like `A1 : C3` for the current sheet containing the button. To reference a different sheet, include a sheet reference like `Sheet3!A1 : C3` to specify cells on a specific sheet.
- ColDelimiter can be `COMMA`, `TAB` or `SEMICOLON` or special characters.
- RowDelimiter can be `LF`, `CR` or `CRLF` or special characters.
- AddQuotes can be `DQ` for double quotes, `SQ` for single quotes, `NQ` for no quotes or special characters.

The Plugins folder is commonly used to share data to and from other applications, a human or a website. Use command `ImportDataFromPlugin` or `ExportDataToPlugin` to read or write data to a named file in the Plugins folder.

Platform Specific

ExcelRT will only execute some commands when running on a specific OS, otherwise the command is ignored.

Command `ImportDataFromMacFile` and `ExportDataToMacFile` will only execute on a Mac computer. Likewise, command `ExportDataToWinFile` will only execute on a Windows computer since those command have OS specific file paths. Command `ExportDataToLinuxFile` will only execute on a Linux computer.

This command is only applicable to iOS and ignored on desktop platforms.

`ShareTextFile(FileName)`

It presents the sharing panel and allows text from the `FileName` referenced in the `Plugins` folder to be shared with other applications. It allows the user to create a CSV text file held within the iOS Files app. The user could email that CSV file to another computer so data could be presented within Microsoft Excel.

These commands are only available on desktop platforms and ignored on iOS. A dialog is presented to pick the font, text color or background color of the specified cell. The `SheetID`, `Col` and `Row` parameters are integers or variables containing an integer value. The first sheet is `SheetID 1`.

- `CellFontPick(SheetID,Col,Row)`
- `CellTextColorPick(SheetID,Col,Row)`
- `CellBackColorPick(SheetID,Col,Row)`

Shared Ticket Folder

The Plugins folder is generally used to exchange data files with other applications. If ExcelRT is installed independent of your application, it may not be installed in the default location. If an external application needs to exchange data with ExcelRT using a known fixed file location, the shared Ticket folder can be used.

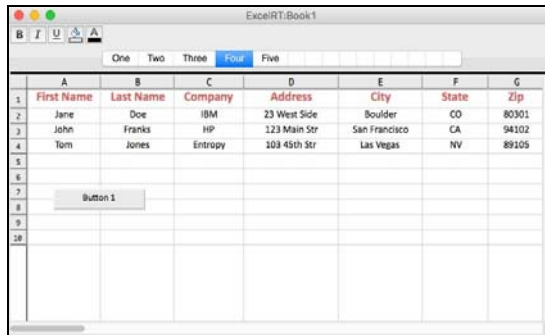
If you use QuickLicense to protect and license your ExcelRT file, you'll be familiar with the shared Ticket folder that holds your active license information. Since all user accounts have full read/write access to this folder. The shared ticket folder is at the same file path on all Mac OS computers, Windows 7 thru 10 computers or Linux computers. It can be a convenient place to read and write data.

Command `ImportDataFromTicketFolder` or `ExportDataToTicketFolder` can read or write data to a named file in the Ticket folder.

- On Windows: `c:\users\public\ticket`
- On MacOS: `/users/shared/ticket`
- On Linux: `/var/ticket`

To demonstrate an export command, consider the cell range `A1:G4` on sheet `Four` displayed in ExcelRT. Assign an export command to `Button 1`.

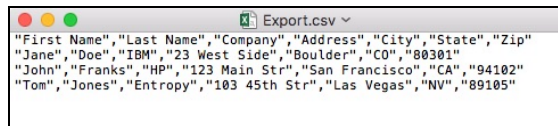
```
Button 1=ExportDataToTicketFolder(Four!A1:G4,COMMA,LF,DQ,"Export.csv")
```



	A	B	C	D	E	F	G
1	First Name	Last Name	Company	Address	City	State	Zip
2	Jane	Doe	IBM	23 West Side	Boulder	CO	80301
3	John	Franks	HP	123 Main Str	San Francisco	CA	94102
4	Tom	Jones	Entropy	103 45th Str	Las Vegas	NV	89105
5							
6							
7							
8							
9							
10							

Export Cell Data to a Text File

Click `Button 1`, then locate file `Export.csv` created in the Ticket folder. Open that file into a plain text editor and it looks something like this.



```
"First Name","Last Name","Company","Address","City","State","Zip"
"Jane","Doe","IBM","23 West Side","Boulder","CO","80301"
"John","Franks","HP","123 Main Str","San Francisco","CA","94102"
"Tom","Jones","Entropy","103 45th Str","Las Vegas","NV","89105"
```

Text File of Cell Data

The `.csv` file extension (comma separated values) is likely mapped to Microsoft Excel. If you double-click that file, it opens Excel and displays the data.

1	First Name	Last Name	Company	Address	City	State	Zip
2	Jane	Doe	IBM	23 West Side	Boulder	CO	80301
3	John	Franks	HP	123 Main Str	San Francisco	CA	94102
4	Tom	Jones	Entropy	103 45th Str	Las Vegas	NV	89105
5							
6							
7							

Data Exported from ExcelRT to Microsoft Excel

Custom Images

To replace a picture in the ExcelRT workbook, identify the sheet title and picture name, then add a command that gets executed by an OnOpen event or button click.

```
ReplacePictureFromPlugin(SheetName,PictureName,FileName)
```

Assume that `Logo.jpg` is stored in the Plugins folder and you want to replace `Picture 1` on `Sheet3` when the workbook opens. See the event actions section below.

```
OnOpen=ReplacePictureFromPlugin("Sheet3","Picture 1","Logo.jpg")
```

To ensure the picture is replaced with a new picture of the same size and position, the replacement picture on disk can be a PNG or JPG image. The picture dimensions on disk must be a specific size based on the original picture pasted into the workbook assuming it has not be scaled within the workbook.

Assume the original picture pasted into the Excel workbook was 470 x 227 pixels. The replacement picture must be 313 x 151 pixels. Multiple the Width and Height by 0.666 to determine the new required Width and Height.

The calculation accounts for the conversion dpi from Excel, the native 96 dpi for Windows and 72 dpi for Mac and the HiDpi display handling within ExcelRT where this calculation is used $X / 2 * 96 / 72$.

This command prompts the user for a file and copies it to the Plugins folder with a specified `FileName`. Apply a company logo to sheets when combined with the `ReplacePictureFromPlugin` command. If the `Message` is not "", it presents a Yes/No dialog to the user giving them the ability to cancel.

```
PluginFileFromUserPrompt(Message,FileName)
```

Internet Data

This command retrieves a file from the Internet and writes it to the Plugins folder with the specified `FileName`.

```
PluginFileFromURL(FileName,URL)
```

To present a URL in the default web browser, use `ShowURL (URL)` as illustrated below.

```
ShowURL("http://www.excelsoftware.com")
```

This command is similar except it presents the URL in a Window owned by the ExcelRT application that can be given a `Title`. The optional `Size` and `Position` parameters allow control over the original size and position of that window.

```
ShowURLWindow(Title,URL,Size*,Position*,BrowserID*)
```

The `Size` parameter can be empty, `Maximize`, `Content` or `WxH` where `W` is a pixel width and `H` is a pixel height. If `Size` parameter is `Content`, the dialog is 75% of the Window size. The `Position` parameter can be empty, `TopLeft`, `Center`, `TxL` or `RelativeTxL` where `T` represents Top position in pixels and `L` represents Left position in pixels. If you supply a `Position` parameter you must at least include an empty `Size` parameter.

The `RelativeTxL` is a relative top and left position from the top left corner of the ExcelRT window itself rather than the top left corner of the screen. Here is a valid example where the presented Browser window is offset down 100 and left 200 pixels from the top left corner of the ExcelRT window.

```
Relative100x200
```

The optional `BrowserID` field returns a named reference to the presented window that can be used to close or interact with controls.

This command closes the window with the assigned `BrowserID`.

```
URLWindowClose(BrowserID)
```

This command refreshes the web page displayed in the window with the assigned `BrowserID`.

```
URLWindowRefresh(BrowserID)
```

This command brings the window with the assigned `BrowserID` to the front.

```
URLWindowFront(BrowserID)
```

This command clicks a control in the Window with the assigned `BrowserID`. `RefType` is id, name, class or tag. `Occurrence` is an integer starting from 1. The `Occurrence` parameter is required, but ignored for `RefType` of id.

```
URLWindowClick(BrowserID,RefType,Occurrence,ID)
```

This command writes text to a control in the window with the assigned BrowserID.

```
URLWindowWrite(BrowserID,RefType,Occurrence,ID,Text)
```

This command copies text from a control in the window with the assigned BrowserID.

```
URLWindowCopy(BrowserID,RefType,Occurrence,ID)
```

This command pastes text to a control in the window with the assigned BrowserID.

```
URLWindowPaste(BrowserID,RefType,Occurrence,ID)
```

This command presents an ExcelRT window to display HTML formatted text. It could be used to display an HTML formatted report constructed by the script without retrieving a URL from the Internet. Title is a variable or literal string containing a title for the window or view. Source contains the HTML formatted text.

```
ShowHTML(Title,Source,Reference,Size,Position)
```

The Reference parameter can usually be left empty. On desktop platforms if your HTML source contains references to image files included in the Plugins folder, then Reference is the name of any file in the Plugins folder.

Reference can also be a full platform specific file path to a file on disk. All referenced files are assumed to be in the folder holding that file. This approach is not recommended since your application would be platform specific.

The Size and Position parameters are optional and work just like those in the ShowURLWindow described above.

Use this command to retrieve data from a web page. URL is a variable or literal string containing the URL of the web page. Result is a variable where the retrieved data is stored.

```
HttpGet(URL,Result)
```

Use this command to post data to a web page. URL is a variable or literal string containing the URL of the web page. postData is a variable or literal string containing field and value pairs to be posted to that page. Result is a variable where the retrieved data is stored.

```
HttpPost(URL,PostData,Result)
```

Here is an example that posts John and Doe to parameters named first and last.

```
HttpPost(http://domain.com/test.php,first=John&last=Doe,Response)
```

Dialog with Button Actions

To show a message dialog with buttons and then perform different actions based on which button is clicked, use the `Message` command. All parameters are strings, where `Message` is the displayed text, `Explanation` is additional text, `Btn1Name` is the default button name, `Btn2Name` is an optional button name or "" and `Btn3Name` is another optional button name.

To perform an action if `Btn1Name` is clicked, then name `Btn1Event` and add an event line to the Button Actions dialog, otherwise use "".

The `Icon` parameter determines what platform specific icon is displayed in the dialog. Use `None`, `Note`, `Caution`, `Stop` or `Question`.

```
Message(Message,Explanation,Btn1Name,Btn2Name,Btn3Name,Btn1Event,Btn2Event,Btn3Event,Icon)
```

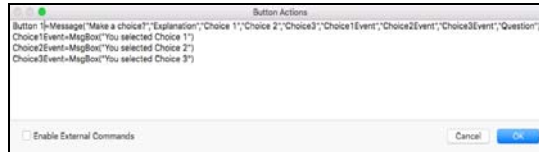
As an example, these commands defined in the Button Actions dialog determine what happens when a button named `Button 1` is clicked.

```
Button 1=Message(Make a choice?,,Choice 1,Choice 2,,Choice1Event,Choice2Event,,Question)
```

```
Choice1Event=MsgBox(You selected Choice 1)
```

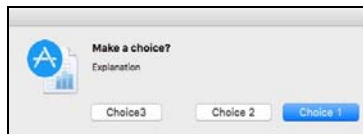
```
Choice2Event=MsgBox(You selected Choice 2)
```

Here are the commands defined in the Button Actions dialog.



Present Message Dialog and Specify Button Actions

Here is the dialog presented when the user clicks `Button 1`.



Choice Dialog Presented to User

External Applications

The folder holding the ExcelRT application contains a `Plugins` folder. In Design mode, the `Plugins` folder holds a text file containing button actions. It can also hold support applications used by ExcelRT in Design or User mode.

To run an application that resides in the Plugins folder, use a platform specific command that is ignored by ExcelRT when running on a different platform.

```
RunMacAppFromPlugin("AppName.app")
RunWinAppFromPlugin("AppName.exe")
RunLinuxAppFromPlugin("AppName")
```

The Plugins folder includes a simple test application `Hello.app` or `Hello.exe`. To try this feature, assign this command to a button and click it.

To run an application that resides at a specified file path, use these commands.

```
RunMacAppFromPath(Path)
RunWinAppFromPath(Path)
RunLinuxAppFromPath(Path)
```

Clipboard and File Data

To write a Data string to the clipboard, use `ToClipboard(Data)` . To read clipboard text and store it in a specified cell, use `FromClipboard(A3)` . These commands can also use variables as described in a later section of this chapter.

To write text to a file, use `ToMacFile(Path,Data)`, `ToWinFile(Path,Data)`, `ToLinuxFile(Path,Data)` or `ToTicketFolder(FileName,Data)`.

A button can run multiple commands separated by the `|` character. Assume you want to pass 2 parameters to an application through the clipboard and then launch an application that can read and use those parameters. Here is an example that uses an application to look up an account balance for John Doe and do something with it.

```
ToClipboard("John Doe")|RunMacAppFromPlugin("AccountBalance.app")
```

Prompt for Data Entry

A button click can present a Data Form dialog to the user to enter data into a sheet. Use `DataForm(ColRange,RowCount,DialogLabel)` where `ColRange` references a cell range that contains field labels, `RowCount` determines the maximum number of rows and `DialogLabel` is a text string that names the dialog.

Assume these cells have label names A1="First Name", B1="Last Name", C1="Company", D1="Street", E1="City", F1="State", G1="Zip". Here is an example of the DataForm command.

DataForm(A1:G1,9,"Company Contact")

First Name	Last Name	Company	Address	City	State	Zip
Jane	Doe	IBM	23 West Side	Boulder	CO	80301

Data Form Dialog for Entering Cell Values

Refer to the ExcelRT window image above where row 1 contains the title of each column of data. If Button 1 is assigned to the DataForm command described above, the Data Form dialog is presented when clicked.

The ColRange parameter in the DataForm command defines the label for each column of data in one row of the table. The first record of data starts in row 2 and a total of 9 rows of data can be created with the Data Form dialog. Notice the 1st of 3 currently defined records is displayed in the dialog. Data is read from the sheet when the dialog is presented and saved to the sheet when the user clicks **OK**.

To prompt for one or more values and store the data in designated cells, use PromptForValue("DialogLabel", "Label1", CellRef1, "Label2", CellRef2, ...). Here is an example.

PromptForValue("Enter Your Name", "First Name", A1, "Last Name", A2)

This command can include any number of label and cell reference pairs. It presents a simplified version of the Data Form dialog that allows user editing of any arbitrary set of cells.

Variables

An ExcelRT file can hold a list of named variables. A variable is usually created with a script command. Local variables are only retained when closing the ExcelRT file by using the `StoreLocalVars` command discussed below.

Assume that you want to prompt the user for text such as a Company name that will be displayed on each sheet of the ExcelRT file. These commands will collect text from the user, assign that text to a variable named `Company` and display it in cell `A1` of the current sheet.

```
PromptUserForVar("Enter Company Name",Company)
VarToCell(Company,A1)
```

The `VarToCell` command puts the value of the named variable into a cell specified with a cell reference. To put a value into a range of cells, use this command.

```
VarToCellRange(VarName,Range)
```

The `VarName` parameter can be a named variable or just data as illustrated in this example that stores the value `0` into 6 cells on `MySheet`.

```
VarToCellRange(0,MySheet!A1:C2)
```

To get text from a cell and store it as a named variable, use this command.

```
VarFromCell(Company,A1)
```

Sometimes when retrieving a value from a cell, you need to use a marker other than an empty string to indicate the empty cell value. This command returns the value `EmptyData` when the cell value is empty. The `VarToCell` command will automatically replace the value `EmptyData` with an empty string.

```
VarFromCellWithEmpty(Company,A1)
```

The `DefineVar` command will define a new variable and give it a default value. This command can also be used to change the value of an existing named variable.

```
DefineVar(VarName,Value)
```

A variable name may contain letters `A..Z`, `a..z` and `0..9`. Other characters should not be used in a variable name. Normal variables are stored in memory and discarded when the ExcelRT file is closed.

To create a global variable that is available to all ExcelRT files, even those created by other products or vendors, start the variable name with `$`. Here is an example of a local and global variable name.

```
DefineVar(Test,Local Data)
DefineVar($Test,Global Data)
```

This command defines a new variable and gives it a literal value. Unlike `DefineVar`, it does not treat the `Value` parameter as a potential variable name.

```
DefineVarLiteral(VarName,Value)
```

This command puts the value of the named variable into a cell specified by SheetID, Col and Row.

```
VarToSheetColRow(VarName,SheetID,Col,Row)
```

This command sets the value of the named variable from a cell specified by SheetID, Col and Row.

```
VarFromSheetColRow(VarName,SheetID,Col,Row)
```

Global variable names and values are retained when you close and reopen the ExcelRT file. Global variables can even be used to exchange data between different ExcelRT files.

To define a variable based on the value in another variable use the DefineVarForValue command. VarName is the variable that you will assign a value to. Test is the name of a test variable. If the Test variable contains Value1 then put Name1 in VarName. If the Test variable contains Value2 then put Name2 in VarName.

This command can include as many Value/Name pairs as desired.

```
DefineVarForValue(VarName,Test,Value1,Name1,Value2,Name2,Value3,Name3,...)
```

Here is an example that assigns the string Female or Male to the Gender variable based on the state of the IsMale variable.

```
DefineVarForValue (Gender,IsMale,FALSE,Female,TRUE,Male)
```

To clear all variable names and values, use this command:

```
ClearVars()
```

Unlike DefineVar, the CreateVar command will only create a named variable with an empty value if it does not already exist. If the variable exists, nothing happens.

```
CreateVar(VarName)
```

This command returns the value of a variable whose name is stored in a variable. Assume Var1 contains the value Var2 and Var2 contains the value Test. This command stores the value Test in a variable named Result.

```
ValueOfVarRef(Var1,Result)
```

This command copies the value from one named variable to another.

```
CopyVar(FromVar,ToVar)
```

Local variable names and values are not saved and restored with an ExcelRT file by default. To force ExcelRT to save local variables so they exist when the file is later reopened, use this command. Carefully read the details below when using this command.

```
StoreLocalVars()
```

When using the StoreLocalVars() command, you will likely want to call it from a script that runs during the OnOpen event, otherwise if the user simply opens and saves the file, the variables are now lost.

Use of the StoreLocalVars command presents the risk of corrupting your ExcelRT file depending on the values stored in variables. An ExcelRT XML file is stored as an ASCII XML formatted text file. An ERT file is just a compressed, encrypted copy of that XML file.

There are scripting commands that allow a developer to store non-ASCII text, prompt the user for text or even store XML tag names in string variables. These actions could damage the structure of the XML file so it cannot be reopened.

A developer can add variables from the Variables dialog presented by clicking the **Variables** tool in ExcelRT Builder. The Variables dialog shows local variable names and values when debugging a script. Global variables discussed below are not displayed in the Variables dialog. This feature could be used to predefine variable values for a specific customer.

Prompt User

Several commands are available to prompt the user for data entry or selections.

A user typically clicks a button on a sheet to enter a variable value into a presented dialog.

This command presents a dialog prompting the user to enter data that is stored in a variable.

PromptUserForVar(Message,VarName)

This command will prompt the user to make a selection from a list of choices, then return a value into a variable based on that choice.

PromptUserForSelection(Description,C1;C2;C3;V1;V2;V3;VarName)

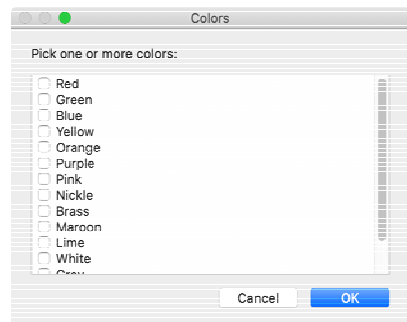
The command has four comma-separated parameters, a description, a choice list, a value list and a variable name.

Use semicolons to separate items in the choice and value list. This example asks the user to select a color.

The PromptCheckboxList command presents a checkbox list in a resizable dialog. List is a variable that holds a semicolon-separate list of items.

Name the dialog the dialog with the Title parameter and give it a one-line description with the Description parameter.

The user can scroll through the list of checkbox, resize the dialog or check multiple items. SelectionVar is a variable that holds a user selected semi-colon separated list of indexes.



PromptCheckboxList(Title,Description,List,SelectionVar)

For ExcelRT Cloud, list items are not shown with a checkbox. The user can hold down the command key while clicking to select one or more non-contiguous items.

Cell References

Sometimes it is useful to construct, deconstruct or offset a cell reference to a sheet of the workbook from other scripting commands that use variables. Consider the cell reference One!B3. It references the cell in column B of row 3 on a sheet titled One.

This command creates a cell reference stored in a variable named CellRef from the Sheet ID, Column number and Row number. The SheetID, Col and Row parameters are integers or variables with an integer value. For example, if the first sheet is titled Data and SheetID is 1, Col is 3, Row is 4, then the CellRef variable will contain Data!C4. FromCol and FromRow are optional parameters that allow you to construct a range in the returned CellRef parameter.

```
ToCellRef(SheetID,Col,Row,CellRef,FromCol*,FromRow*)
```

This command returns the Sheet ID, Column number and Row number from a cell reference. The parameter CellRef can be a literal string or variable name holding the cell reference. For example, if the title of the second sheet is Options and CellRef contains Options!D7, this command will return SheetID 2, Col 4 and Row 7.

```
FromCellRef(SheetID,Col,Row,CellRef)
```

This command applies a Sheet, Col or Row offset to the cell reference in the named CellRef variable. For example, if the cell reference is One!B5 and Sheet is 0, Col is 0 and Row is 1, then the cell reference is changed to One!B6. Exceeding the Sheet, Col or Row boundaries will cause an exception.

```
OffsetCellRef(Sheet,Col,Row,CellRef)
```

This command is given a list of cell references and returns the first empty cell reference into the Result variable.

```
FindFirstEmptyCell(CellRef1,CellRef2,CellRef3,...,Result)
```

Assume that you want to collect 20 sets of similar data from a user. A custom dialog presented with a button click collects all the data for one set of cells. The FindFirstEmptyCell command can locate the first empty set of data into which the dialog results are stored.

Math

Use the Math command to perform basic math between two Operands and stores the output in a Result variable. Operand1 and Operand2 can be a literal value or variable name. The Operator parameter can be +, -, / or *.

```
Math(Operand1,Operator,Operand1,Result)
```

Use this command to access the calculation engine built into ExcelRT from the scripting language. The Formula parameter is a formula from a variable or literal string. The same syntax is supported as that used in any cell within the workbook without the leading = character. The calculated result is stored in the Result parameter.

```
Calculate(Formula,Result)
```

Here is an example of the Calculate command. If the formula includes any commas you will need double quotes around it when entering it as a literal string.

```
Calculate("1+Sin(A4)",MyAnswer)
```

This command returns a random integer value between the From and To value.

```
Random(From,To,Result)
```

Lists

These commands can manipulate a list of values. Source is a delimiter-separated list of string values or named variable containing such a list. Delimiter can be the actual delimiter or named variable that contains the delimiter character. Index is an integer value or named variable that contains an integer.

The output of the command is stored in the named variable identified by the Result parameter. The SetNthField command changes the value of the Source variable.

```
NthField(Source,Delimiter,Index,Result)
CountFields(Source,Delimiter,Result)
FindField(Source,Delimiter,Find,Result)
SetNthField(Source,Delimiter,Index,NewValue)
DefineFields(Source,Delimiter,Default,Count)
AddField(Source,Delimiter,NewValue)
DeleteNthField(Source,Delimiter,Index)
SortFields(Source,Delimiter,CaseSensitive)
ReverseFields(Source,Delimiter)
WhileFields(Source,Delimiter,FieldVar,Sub)
```

These commands define a variable named Var1 containing three semicolon-separated strings, then retrieves the second string into Var2 and present it in a dialog.

```
DefineVar(Var1,One;Two;Three)|NthField(Var1,;,2,Var2)|MsgBox(Var2)
```

These commands count the strings in Var1 and display the count in a dialog.

```
CountFields(Var1,;,Var2)|MsgBox(Var2)
```

These commands find and display the index of a string `Three`.

```
FindField(Var1,,Three,Var2)|MsgBox(Var2)
```

This command changes the value of the second string and displays the list.

```
SetNthField(Var1,,2,Hello)|MsgBox(Var1)
```

The `DefineFields` command creates a delimited list with `Count` elements each containing the `Default` string. The `Source` parameter is a named variable, while other parameters can be a literal string or named variable.

```
DefineFields(MyArray,,0,10)
```

A delimited list can be used like an array where `NthField` is used to get an array value and `SetNthField` is used to set an array value.

The `AddField` command adds a new field to the end of the list. The `Source` parameter is a named variable. The `Delimiter` and `NewValue` parameters can be a named variable or literal string.

```
AddField(Source,Delimiter,NewValue)
```

Use `AddUniqueField` to add `NewValue` if it doesn't already exist in the `Source`.

```
AddUniqueField(Source,Delimiter,NewValue)
```

The `DeleteNthField` command removes the item at the specified `Index`. The first item is at index 1.

```
DeleteNthField(Source,Delimiter,Index)
```

The `SortFields` command will alphabetically sort the strings in the item. The `CaseSensitive` parameter should be `TRUE` or `FALSE`.

```
SortFields(Source,Delimiter,CaseSensitive)
```

This command reverses the order of delimited strings stored in the `Source` variable:

```
ReverseFields(Source,Delimiter)
```

Use this command to create a delimited list of strings from values in a range of cells.

```
FieldsFromCellRange(Source,Delimiter,Range)
```

Use this command to store a delimited list of strings into a range of cells.

```
FieldsToCellRange(Source,Delimiter,Range)
```

A list of semicolon-separated strings can be retrieved from a range of cells and then presented to the user with the `PromptUserForSelection` or `Dialog` commands.

Use this command to loop through the field list. For each loop iteration, the value of the field is stored `FieldVar` and the named Subroutine is called.

```
WhileFields(Source,Delimiter,FieldVar,Sub)
```

Email Read

This command allows a workbook to receive information from a specific email address by requesting the email message directly from an email server.

EmailRead(Host,Port,User>Password,Type.SLL,Format,ResultVar,ReasonVar,FromVar,ToVar,SubjectVar,MessageVar)

These are input parameters:

- Host is the email server (example: pop.secureserver.net)
- Port is the port number (example: 110)
- User is the username of account (example: tom@yourdomain.com)
- Password is the incoming email account password (example: test123)
- Type is FALSE for Pop2 or TRUE for IMAP
- SSL is FALSE or TRUE
- Format is ASCII, ASCII7, ASCII8, UNICODE or RAW

The return parameters are variable names:

- ResultVar is SUCCESS, ERROR or FAILED
- ReasonVar contains error description if not SUCCESS
- FromVar contains the sender email address
- ToVar contains the receiver email address
- SubjectVar contains the Subject line text
- MessageVar contains the message content based on Format parameter

External App Communication

Several commands can accept a variety of parameter types for the VarName field including a variable name, CellRef or Data. These commands are typically used to communicate data between ExcelRT and an external helper application.

- ToPluginFile(Filename,VarName)
- FromPluginFile(Filename,VarName)
- ToClipboard(VarName)
- FromClipboard(VarName)

For example ToPluginFile and ToClipboard writes text to a file in the Plugins folder or to the clipboard, respectively. ExcelRT firsts checks if VarName is a defined local or shared variable, then checks if it is a valid CellRef otherwise it is assumed to be a literal text string.

A CellRef identifies a specific cell in the local sheet like B7 or a cell on a specific sheet like ConfigSheet!D2.

This command defines a local or shared variable. To create a shared variable, start the name with a \$ character. A variable is typically defined prior to a command that reads data into the named variable.

- DefineVar(VarName,Data)

These helper functions are often useful when communicating with an external application. The Sleep command waits the specified number of seconds, usually to allow a helper application to

collect some data. The `Seconds` parameter can be a value from 1 to 9. The `PluginFileDelete` command deletes the specified `FileName` in the `Plugins` folder.

- `Sleep(Seconds)`
- `PluginFileDelete(FileName)`
- `PluginWaitForFile(FileName,Timeout)`

The `PluginWaitForFile` command periodically polls for the existence of the specified `FileName` in the `Plugins` folder. It will time out if that file is not found for `Timeout` seconds.

Picture and PDF Files

Use these command to present an image or PDF file stored in the Plugins folder. The Title parameter titles the presented window. The FileName parameter is the file name without the path in the Plugins folder. Refer to the ShowURLWindow command for details regarding the Size and Position parameters.

- ShowPictureFromPlugin(Title,FileName,Size*,Position*)
- ShowPDFFromPlugin(Title,FileName)

On a desktop platform, the FilePath parameter specifies a platform specific full file path to the image or PDF file that can reside anywhere on disk.

- ShowPictureFromPath(Title,FilePath,Size*,Position*)
- ShowPDFFromPath(Title,FilePath)

For compatibility with future ExcelRT Cloud, the ShowPDFFromPlugin and ShowPDFFromPath command may include Size and Position parameters, but they are ignored in a Desktop edition of ExcelRT.

The ShowPDFFromPlugin and ShowPDFFromPath commands can also be used to show a video or other document type on most platforms. To use this feature, test this command with your specific document on each ExcelRT platform that your product claims to support.

This command creates a JPG image file in the Plugins folder for the current sheet.

```
SheetToJPG(Filename,MaxColumn*,MaxRow*)
```

MaxColumn and MaxRow are optional parameters that allow you to limit the JPG size to the specified number of columns and rows. This command creates a JPG in the Plugins folder named Image.jpg from current sheet cells A1:D5.

```
SheetToJPG(Image.jpg,4,5)
```

This command creates a JPG image file in the Plugins folder for the named sheet.

```
SheetNameToJPG(SheetName,Filename,MaxColumn*,MaxRow*)
```

This command creates a PDF image file in the Plugins folder for the current sheet.

```
SheetToPDF(FileName,MaxCol*,MaxRow*)
```

In this example, the generated image is stored in file Sheet1.pdf and has 5 columns and 10 rows

```
SheetToPDF(Sheet1.pdf,5,10)
```

This command creates a PDF image file in the Plugins folder from a JPG file. The optional PageSize parameter can be Letter, Legal, Tabloid, A3, A4 or Default. Default will size the PDF document to the image size it contains.

```
JPGtoPDF(File.jpg,File.pdf,PageSize*,Title*,Subject*,Author*)
```

The Title, Subject and Author fields are optional searchable meta data within the PDF.

```
JPGtoPDF(File.jpg,File.pdf)
```

This command prints a JPG file from the Plugins folder. The Scale parameter is TRUE or FALSE. When TRUE, the JPG file is scaled to fit the printed page.

```
PrintJPG(Filename,Scale)
```

The Filename parameter in the PrintJPG command can be a semi-colon separated list of file names like `File1.jpg;File2.jpg;File3.jpg`. This may be useful on a desktop computer to print multiple images from a generated report. Those images are combined into the same print job so by printing to a PDF file, they get merged into a single PDF file.

Printing on iOS goes through the sharing panel that only holds one image at a time. On iOS, only the first supplied JPG file is printed.

Subroutines

A script can define or call a subroutine. A subroutine can be used to break up a long list of commands into smaller parts or to create a common list of commands that get called by multiple buttons.

To demonstrate, consider this button action. When the button is clicked, three sequential dialog boxes are presented.

```
Button 1=MsgBox(One)|MsgBox(Two)|MsgBox(Three)
```

Here is another way to accomplish the same result. It uses a subroutine named `Sub$One`. By convention, each subroutine name consists of the prefix `Sub$` plus a unique name.

```
Button 1=Sub$One|MsgBox(Three)
Sub$One=MsgBox(One)|MsgBox(Two)
```

Create as many subroutines as needed. A subroutine can call other subroutines as illustrated below. When the button is clicked, five sequential dialogs are presented. Be careful not to create a circular loop between subroutine calls.

```
Button 1= Sub$1|MsgBox(Five)
Sub$1=Sub$2|Sub$3
Sub$2=MsgBox(One)|MsgBox(Two)
Sub$3=MsgBox(Three)|MsgBox(Four)
```

Conditionals and Loops

ExcelIRT supports conditional logic. These commands set a named result variable to TRUE or FALSE based on the values in a list of named variables.

```
AndSetVar(Result,Var1,Var2,...)
OrSetVar(Result,Var1,Var2,...)
```

This command sets a named Result variable to TRUE or FALSE based on a condition between two operands. Operand1 and Operand2 can be a literal value, named variable or supported symbol.

```
IfSetVar(Operand1,Condition,Operand1,Result,CompareType*,Value*)
```

The Condition parameter can be =,<>,<,<=,> or >=.

Supported symbol names are shown in the Symbols section below. Most symbols are available from an XML or ERT file at any time. Some symbols like `SerialNumber` are only valid under specific conditions. For example, your Product must be activated with a Serial Number before that symbol returns a non-empty value.

In the first example, the variable named `FirstHalf` is set to TRUE if the current date is in the first half of the year and FALSE for the second half. The second line sets the variable named `IsMac` to TRUE if ExcelIRT is running on a Mac.

```
IfSetVar(Month,<=,June,FirstHalf)
IfSetVar(Target,=,Mac,IsMac)
```

The `CompareType` and `Value` parameters in the `IfSetVar` command are optional. Use `Number`, `String` or `StringCase` for the `CompareType` parameter where `Number` assumes the operands are a number, `String` assumes a case insensitive string comparison and `StringCase` is a case-sensitive comparison. `Value` is a literal or variable value that is assigned to `Result` if the condition is TRUE.

This command sets a named variable to the value of a symbol listed above.

```
SymbolValue(VarName,Symbol)
```

This command runs a named set of commands (subroutine) based on the TRUE or FALSE result of the condition between two operands. Each operand can be a literal value, named variable or symbol as described above.

CompareType can be Default, Number, String or StringCase to indicate how operators should be treated when doing a comparison. If no CompareType parameter is included (default behavior of ExcelRT 1.0), then Default is assumed. For Default, = and <> treats operators as String while other conditions assume operators are Number.

```
IfDoEvent(Operand1,Condition,Operand1,trueEvent,falseEvent,CompareType*)
```

For best results, use Number, String or StringCase for the CompareType parameter where Number assumes the operands are a number, String assumes a case insensitive string comparison and StringCase is a case-sensitive comparison.

This command runs Sub\$One if MyName is Tom, otherwise it runs Sub\$Two.

```
IfDoEvent(MyName,=,Tom,Sub$One,Sub$Two)
Sub$One=MsgBox(My name is Tom)
Sub$Two=MsgBox(My name is not Tom)
```

This command runs the named Sub if the Operand is TRUE.

```
IfEvent(Operand,Sub)
```

The SelectEvent command is like a switch statement in that it compares the value of the TestVar with one or more literal or variable values. If a match is found, it runs the associate subroutine. This command can use named symbols.

```
SelectEvent(TestVar,Var1,Sub1,Var2,Sub2,Var3,Sub3,...)
```

In this example, a dialog indicates on which platform this script is running.

```
SelectEvent(Target,Mac,Sub$Mac,Windows,Sub$Win,iOS,Sub$iOS)
Sub$Mac=MsgBox(Mac)
Sub$Win=MsgBox(Window)
Sub$iOS=MsgBox(iOS)
```

The While statement creates a loop structure that can count up or down and call the referenced subroutine. The First and Last parameters can be an integer value or named variable that contains an integer value.

```
While(First,Last,Sub,IndexVar*)
```

This example, displays a message dialog three times.

```
While(1,3,Sub$One)
Sub$One=MsgBox(Loop1)
```

Since one subroutine can call another, you can create nested While loops as illustrated below.

```
While(1,3,Sub$One)
Sub$One=MsgBox(Loop1)|While(2,4,Sub$Two)
Sub$Two=MsgBox(Loop2)
```

The fourth optional parameter in this While loop holds an Index value presented in a MsgBox as 1, 2 and 3.

```
While(1,3,Sub$One,Index)
Sub$One=MsgBox(Index)
```

This command calls one of three Subs based on a numeric or string comparison of two values. Type is Number, String or StringCase for a numeric, string or case sensitive string comparison.

```
LessEqualGreater(TestValue,CompareValue,Type,LessSub,EqualSub,GreaterSub)
```

This command calls one of three Subs based on a numeric or string comparison of two values. Type is Number, String or StringCase for a numeric, string or case sensitive string comparison. If the Value is equal to or between the Low and High limits, then call RangeSub.

```
LessRangeGreater(Value,Low,High,CompareType,LessSub,RangeSub,GreaterSub)
```

Symbols

Supported symbol names are shown below.

Target	Mac	Windows	Linux
IOS	Android	Day	Month
Year	Hour	Minute	Second
DayOfWeek	DayOfYear	WeekOfYear	Today
Time	TimeStamp	Sunday	Monday
Tuesday	Wednesday	Thursday	Friday
Saturday	January	February	March
April	May	June	July
August	September	October	November
December	SheetID	ScreenWidth	ScreenHeight
Filename	Device	Computer	Phone
Tablet	MessageState	BaseFileName	CR
LF	CRLF	SelectedRow	SelectedColumn
LastSelectedRow	LastSelectedColumn		ExcelRTVersion
PictureClickX	PictureClickY	Volume	Comma
DoubleQuote	OpenParen	CloseParen	PluginFolder

The Device symbol returns Computer, Phone or Tablet to allow your workbook to alter its behavior on different devices. On iPhone, Target returns iOS and Device returns Phone. On iPad, Target returns iOS and Device returns Tablet.

The CR, LF and CRLF symbols are used to add line endings on text.

The MessageState symbol returns TRUE or FALSE to indicate if the OnMessage event is enabled. This symbol is only applicable for ExcelRT Vendor Commands.

These symbols are only valid for a licensed product. An empty string is returned for a simple XML or ERT file. These symbols work with a Standalone, Shared QuickLicense or Shared Cloud License application.

RequestNumber	SerialNumber	VendorID	Features
ProductID	CloudLicenseStatus		

RequestNumber returns a 10-digit number for the activated computer or device. SerialNumber returns the Serial Number entered by the user to activate the product. VendorID returns your 8-digit VendorID for Safe Activation Service 3 for the activated license. ProductID returns an integer value that represents the Product record in Safe Activation associated with QuickLicense or Cloud License.

CloudLicenseStatus indicates the status of the current license with values NOSETUP, NOTFOUND, UNACTIVATED, ACTIVATED, EXPIRED, BLOCKED or SUSPENDED.

Features returns a 100-character string used to enable or disable up to 100 unique features in your product. The 0 character indicates that feature is disabled and X indicates that feature is enabled. The CardID symbol is used to implement InApp credit card payments within an ExcelRT workbook. The symbol is stored on the local computer or device for a specific VendorID of a Safe Activation Service 3 account.

CardID

Credit card information is securely vaulted in the Paypal payment processor. It is not stored in the ExcelRT file, application or computer or on the Safe Activation server.

The OnPictureClick event is triggered when a named picture control is clicked. The event can run script commands that retrieve the click position within the picture using the PictureClickX and PictureClickY symbols.

Regardless of where a picture is positioned within a sheet or the settings of the horizontal or vertical scrollbars, the top left corner of the picture represents X=1 and Y = 1, while the bottom right corner represents X=Width and Y = Height where Width and Height are the pixel size of the picture control.

CSV Read, Write and Modify

ExcelRT provides a flexible memory structure to read and write CSV formatted data.

Item Name	Dimensions	Part Number	Price
1/4" Plywood	4'x8'	AB2567PL1/4	20.57
1/2" Plywood	4'x8'	AB2567PL1/2	24.92
5/8" Plywood	4'x8'	AB2567PL5/8	27.36
3/4" Plywood	4'x8'	AB2567PL3/4	32.87

A CSV text file contains one or more rows of comma-separated values. A script can read a CSV file into memory, copy individual cells from that memory structure to and from variables and write that memory structure to a text file using the same or different dimensions, column or row separators.

```
CreateCsv(Cols,Rows)
CellRangeFromOpenCsv(Range,FromRow,FromColumn,ToRow,ToColumn)
CellRangeToOpenCsv(Range,FromRow,FromColumn,ToRow,ToColumn)
OpenCsvFromPlugin(ColDelimiter,RowDelimiter,AddQuotes,Filename,Cols,Rows)
SaveCsvToPlugin(ColDelimiter,RowDelimiter,AddQuotes,Filename,Cols,Rows,ForceQuotes*)
VarFromOpenCsv(VarName,Row,Column)
VarFromOpenCsvNumber(VarName,Row,Column)
VarToOpenCsv(VarName,Row,Column)
```

Use this command to create a CSV memory structure with a specified Number of columns and rows. Once a CSV is created, you can add space for additional rows by calling this command again with a larger Rows parameter without destroying existing data.

```
CreateCsv(Cols,Rows)
```

Additional columns cannot be added to an existing CSV without scrambling the data. To clear the CSV data, use this command.

```
CreateCsv(0,0)
```

Refer to section Cell Data Import and Export to understand parameters ColDelimiter, RowDelimiter and AddQuotes. Filename refers to a CSV text file stored in the Plugins folder. The Cols and Rows parameters in the OpenCsvFromPlugin command must exactly match the column count and row count in the CSV formatted data file.

The Row and Column parameters refer to a specific cell in memory. The SaveCsvFromPlugin command can save modified data in memory to the same or different Filename in the Plugins folder. The total cell count based on the Cols and Rows parameters must not exceed the total cell count currently stored in memory. ForceQuotes is an optional parameter that defaults to TRUE. If TRUE, every cell value is quoted based on the AddQuotes parameter, otherwise only those cell values are quoted that need to be.

The command VarFromOpenCsvNumber is similar to VarFromOpenCsv except it strips off any leading \$, space or comma found within the text so it represents a number that can be used in a calculation.

Additional commands are available to query, add, modify or delete CSV memory data. These commands typically assume that the first row of data consists of field names that identify the data in that column.

This command retrieves data from a CSV memory structure based on an exact, non case-sensitive match.

```
CsvQuery(MatchField,MatchValue,ResultDelimiter,StatusVar,ResultVar,Field1,Field2,...)
```

The command assumes that data is already stored in memory. MatchField is the name of the field to search. MatchValue is the value of the named field to match. ResultDelimiter is the delimiter used to separate result values including COMMA or SEMICOLON. StatusVar is a variable that stores the status of the command of SUCCESS, FAILED or NOMATCH. ResultVar is the variable name where the result is stored. FieldX can be one or more named fields separated by commas for which the value is returned in the ResultVar parameter.

This command retrieves a row number from a CSV memory structure into a named variable. The command assumes that CSV data is stored in memory.

```
CsvQueryRow(MatchField,MatchValue,StatusVar,RowVar)
```

This command adds a row of data to the CSV memory structure. The command assumes the CSV memory structure is already created. The number of supplied field values must match the number of columns. Data is inserted at the row number or at the end of the CSV table if the specified row exceeds the last row. The first row starts at 1 whether it starts with a header row or not. Here is an example that adds data to the end of table assuming it has under 9999 rows.

```
CsvAddRow(Row,Field1,Field2,...)
```

This command replaces a row of data in the CSV memory structure. The command assumes the CSV memory structure is already created and the number of supplied field values matches with the number of columns. Data is replaced at the specified row number. The first row starts at 1 whether it starts with a header row or not.

```
CsvReplaceRow(Row,Field1,Field2,...)
```

This command deletes a row of data in the CSV memory structure. The command does nothing if the specified Row is larger than the maximum row. The first row starts at 1 whether it starts with a header row or not.

```
CsvDeleteRow(Row)
```

This command returns the number of rows in the CSV memory structure into the named variable. The first row starts at 1 whether it starts with a header row or not. ColVar is an optional parameters that returns the column count.

```
CsvRowCount(RowVar,ColVar*)
```

This command sorts the rows in the CSV memory structure. SortColumn is the column of data values that determine the sorted order. Direction is UP or DOWN to sort in ascending or descending alphabetical order. Header is TRUE or FALSE to indicate if the first row should be sorted.

```
CsvSort(SortColumn,Direction,Header)
```

This command is used to discover the format and size of a CSV file in the Plugins folder. ColDelimiterVar is a named variable that returns COMMA, SEMICOLON or TAB. RowDelimiterVar returns LF, CR or CRLF. AddQuotesVar returns DQ, SQ or NQ.

```
CsvFormatFromPlugin(Filename,ColDelimiterVar,RowDelimiterVar,AddQuotesVar,  
ColCountVar,RowCountVar,HeaderVar,StatusVar)
```

The number of columns and rows including the header row is returned in ColCountVar and RowCountVar. A comma-separated list of header names from the first row is returned in the HeaderVar variable. The StatusVar variable returns SUCCESS, NOTFOUND or FAILED if the file does not satisfy a supported format.

This command retrieves a column of data from CSV memory structure into a sheet. CsvColumn is the column number of the CSV structure. SheetID identifies target sheet, StartRow indicates the starting row number and SheetColumn indicates the column used to store the data. Make sure sheet has enough rows to hold all data in the CSV.

```
SheetColumnFromOpenCsv(CsvColumn,SheetID,StartRow,SheetColumn)
```

This command retrieves a row of data from CSV memory structure into a sheet. CsvRow is the row number of the CSV structure. SheetID identifies target sheet, SheetRow indicates the number and StartColumn indicates the starting column used to store the data. Make sure sheet has enough columns to hold all data in the CSV.

```
SheetRowFromOpenCsv(CsvRow,SheetID,SheetRow,StartColumn)
```

This command presents the CSV Viewer dialog.

```
CsvShow()
```

This command loads CSV formatted data from a string. The string may come from a nested folder Plugins folder or may have been read from disk and decrypted. It works exactly like the OpenCsvFromPlugin command except data comes from the string rather than a file.

```
OpenCsvFromString(ColDelimiter,RowDelimiter,AddQuotes,StrVar,Cols,Rows)
```

This command stores CSV formatted data from memory to a string. It works exactly like the SaveCsvToPlugin command except data is saved to the string rather than a file.

```
SaveCsvToString(ColDelimiter,RowDelimiter,AddQuotes,StrVar,Cols,Rows,ForceQuotes*)
```

This command creates a JSON formatted string from the active CSV memory structure and stores it in StrVar.

```
CsvToJsonString(StrVar)
```

Multiple CSVs

Initially, ExcelRT only supported a single CSV structure in memory to import and export data. Some applications can be simplified and run much faster by storing multiple CSV structures in memory at once. This allows the application to convert data from one CSV structure to another by adding, removing or reordering columns or rows.

A CSV structure can also be used as a simple database. The workbook can load the CSV on open, query for data and later save the CSV on workbook close if changes are made. CSV files can be stored in the Plugins folder, in a Cloud Sharing account or elsewhere on the Internet.

ExcelRT supports 5 CSV structures. Most CSV related commands use CSV 1 by default. This command assigns all CSV related commands that follow to use 1 of 5 CSV memory structures. The CsvNumber parameter may contain 1 to 5 or a variable that contains the integer value.

```
CsvActive(CsvNumber)
```

For example, after the CsvActive(2) command, the CsvShow() command presents the CSV Viewer dialog with data from CSV 2. All CSV related commands that do not specify the CSV number are affected by the last CsvActive command.

This command copies a value from one CSV to another CSV. All parameters can be an integer or variable containing an integer.

```
CsvCopyValue(FromCsv,FromRow,FromCol,ToCsv,ToRow,ToCol)
```

This command copies a column of values from one CSV to another CSV.

```
CsvCopyColumn(FromCsv,ToCsv,Column)
```

This command copies a row of values from one CSV to another CSV.

```
CsvCopyRow(FromCsv,ToCsv,Row)
```

This command defines an array and copies in a row of values from a CSV. The number of columns in the CSV determines the array size.

```
ArrayFromCsvRow(ArrayName,Csv,Row)
```

This command defines an array and copies in a column of values from a CSV. The number of rows in the CSV will determine the array size.

```
ArrayFromCsvColumn(ArrayName,Csv,Column)
```

This command copies values from an array into one row in a CSV. The number of columns in the CSV should match the array size.

```
ArrayToCsvRow(ArrayName,Csv,Row)
```

This command copies values from an array into one column in a CSV. The number of rows in the CSV should match the array size.

```
ArrayToCsvColumn(ArrayName,Csv,Column)
```

Fuzy Queries

Use this command to query CSV memory data for a fuzy match and return partial match row numbers. Unlike a typical query command, matching data may have extra or missing spaces or characters, phrases in a different order and other variations that may occur when comparing data from different sources. A set of potential matches can be presented to the user to select the desired match. Those user-selected matches can be stored so the system learns over time.

```
CsvFuzyQuery(MatchField,MatchValue,Strategy,StrategyMiss,SynonymArray,ConfirmedMatchArray,MaxMatch,MatchingRowsVar,StatusVar,IgnoreArray*)
```

This command assumes the CSV data includes one row of field names. MatchField can be an integer value indicating the column of CSV data to use as the haystack being searched. If not an integer, MatchField must match one of the field names in the first row of the CSV data. MatchValue is the Needle text to compare against values in the haystack. StatusVar will return NOMATCH, PARTIALMATCH, FULLMATCH or FAILED.

If status is PARTIALMATCH or FULLMATCH, then MatchingRowsVar contains one or more semicolon separated row numbers. Use these row numbers to retrieve values from appropriate column of those rows to present a dialog to the user. When the user selects the desired match, add to the ConfirmedMatchArray and retrieve other fields from that CSV file based on the selected row.

Strategy

Strategy is a string containing identifiers that indicate which Strategies to use to find a match. For example, assume several strategies are implemented. To use strategies 1 and 2, supply the Strategy parameter value 12. To explain each strategy, the Needle is what you are searching for and the Haystack is the column of values to be searched. Searches are NOT case sensitive.

The developer can choose the strategy priority. For example, a Strategy value of 132 runs strategy 1, then 3 and then 2. The MaxMatch field allows the designer to terminate the search process early if a specified number of matches are found. If the MaxMatch fields is 9 and 15 partial matches are found, only the first 9 highest priority matches are returned.

Strategy 1 - Character Search

Individually search for each non-space Needle character in the Haystack. If EVERY character is found, it's a match.

Strategy 2 - Reverse Character Search

Individually search for each non-space Haystack character in the Needle. If EVERY character is found, it's a match.

Strategy 3 - Phrase Search

Individually search for each non-space Needle phrase (contiguous characters) in the Haystack. If EVERY phrase is found, it's a match.

Strategy 4 - Reverse Phrase Search

Individually search for each non-space Haystack phrase (contiguous characters) in the Needle. If EVERY phrase is found, it's a match.

Strategy 5 - Partial Character Search

This is similar to the Character Search strategy, except allow X number of missed characters as specified by the StrategyMiss parameter.

Strategy 6 - Partial Reverse Character Search

This is similar to the Reverse Character Search strategy, except allow X number of missed characters as specified by the StrategyMiss parameter.

Strategy 7 - Partial Phrase Search

This is similar to the Phrase Search strategy, except allow X number of missed characters as specified by the StrategyMiss parameter.

Strategy 8 - Partial Reverse Phrase Search

This is similar to the Reverse Phrase Search strategy, except allow X number of missed characters as specified by the StrategyMiss parameter.

Synonym Array

The SynonymArray is the name of an array where each array element consists of one or more semicolon separated synonyms such as 'ft;feet;lf;lf. Prior to performing any search strategies, all SynonymArray values are searched and removed from both the Needle and Haystack value. For a synonym match to occur, the synonym name must have leading and trailing white space unless it is at the beginning or end of the string.

Confirmed Match Array

The ConfirmedMatchArray is the name of an array containing values of the form Item1=MatchItem1 where Item1 is the Needle and MatchingItem1 is a user confirmed partial match from the Haystack. Before considering the SynonymArray or search strategies, if the Needle is found in the ConfirmedMatchArray, then search the Haystack for all MatchItems and return a FULLMATCH if found.

Ignore Array

The optional IgnoreArray can be used to remove a set of words from the Needle and Haystack values before applying synonyms or running search strategies. By removing common brand or descriptive words from a list, a query command can yield better results. The IgnoreArray is typically constructed using the CommonWords command, before using the CsvFuzyQuery command.

This command counts the occurrences of common words that appear in one array but not the other. It returns an array of those common words. Array1 and Array2 are the input arrays. Count determines how many occurrences designate a common word. Array3 contains the output.

CommonWords(Array1,Array2,Count,Array3)

String

This command replaces all occurrences of Find with Replace in Source and stores the output string in Result.

```
ReplaceStrWithStr(Source,Find,Replace,Result)
```

Use this command to put the character length of Source in Result.

```
Length(Source,Result)
```

Use this command to put the character index of Find string starting at 1 into Result. Source contains the string to search, Find contains the string to search for, Start contains the Index from which to begin the search and CaseSensitive contains TRUE or FALSE indicating whether or not to do a case sensitive search.

```
Pos(Source,Find,Start,CaseSensitive,Result)
```

Use this command to copy a portion of the Source parameter to the Result parameter based on a Start index and Length.

```
Mid(Source,Start,Length,Result)
```

To concatenate a variable length list of values into a destination variable use the `Concat` command. DestVar is a variable or CellRef. If it does not identify a named variable or cell reference that exists, an error is presented.

```
Concat(DestVar,Param1,Param2,Param3,...)
```

The remaining parameters can be a variable or CellRef. If none of these exist, then the parameter is assumed to be a literal string. In this example string "My name is John Doe" is constructed and stored in a variable named Message.

```
Concat(Message,"My name is ",FirstName," ",LastName)
```

Use the `FormatStr` command to strip out characters not included in a supplied Format string. If CaseSensitive is FALSE, then ignore case. If ErrorMessage is a non-empty string, then present an error message if one or more characters are filtered out of the Source string.

```
FormatStr(Source,Format,CaseSensitive,ErrorMessage)
```

Here is an example (assume these commands are on one line), that presents a dialog displaying "Tom" from the original string of "1;-&'To2m".

```
Button 1=DefineVar(FirstName,1;-  
&'To2m)|DefineVar(Filter,abcdefghijklmnopqrstuvwxy)|FormatStr(FirstName,Filter,FALSE,)  
|MsgBox(FirstName)
```

This command presents an error message if any character does not match the filter.

```
FormatStr(FirstName,abcdefghijklmnopqrstuvwxy,FALSE,Enter Alpha characters A..Z)
```

This command encrypts Source string into the Output string using a Password. Source is a variable name or literal text. Output is the name of a variable holding the encrypted string. If either the Source or Password text is under 5 characters, the output string is ERROR.

```
Encrypt(Source,Output>Password)
```

This command decrypts Source string into the Output string using the same Password used to encrypt it. Source is a variable name or literal text. Output is the name of a variable holding the decrypted string.

```
Decrypt(Source,Output>Password)
```

The Encrypt and Decrypt commands can be used to secure any text data including CSV files that you want to distribute by email or on the Internet.

Dialog

The `Dialog` command is used to present a custom dialog with up to 18 controls of type Edit, List, Switch or Caption. Here is the list of parameters.

`Dialog(Title,Btns,DlgResult,Control1,Control2,Control3,...)`

The `Title` parameter names the dialog. The `Btns` parameter may contain 1 to 4 button names separated by semicolons. These buttons are displayed at the bottom of the dialog and will dismiss the dialog. The `DlgResult` parameter is a variable that contains the button name clicked by the user to dismiss the dialog.

Each `Control` parameter can be one of four control types. Notice how the parameters within an individual control are separated by the `#` character.

- `Edit#Label#Result#Default`
- `List#Label#ListNames#ListValues#Result#Default`
- `Switch#Label#Result#Default`
- `Caption#Label#Help`

`Label` is literal text or a named variable holding text. `ListNames` is a semicolon-separated list of strings presented in the `List` control. `ListValues` is a semicolon-separated list of strings returned when a user selects a list item. `Result` is the name of the variable holding the user selection. For a `Switch` control, `TRUE` or `FALSE` is stored in `Result`.

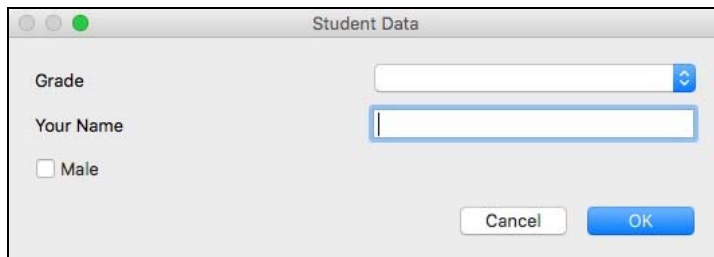
`Default` is an optional parameter. For an `Edit` control it provides text that is initially stored in the edit field. For a `List` control, `Default` is an integer starting with 1 that contains the default selection. For a `Switch` control, `Default` is `TRUE` or `FALSE`.

Here is an example script (must be on one line) that presents a dialog when the user clicks on **Button 1**.

That dialog contains a `List`, an `Edit` field and a `Switch` plus **Cancel** and **OK** buttons at the bottom that dismiss the dialog.

```
Button 1=Dialog(Student Data,Cancel;OK,DlgResult,List#Grade#First;Second;Third#1;2;3#Grade,Edit#Your Name#Name,Switch#Male#Gender)
```

When the user clicks **Button 1**, the dialog titled "Student Data" is presented. The user can select First, Second or Third to store 1, 2 or 3 into the `Grade` variable, type a name to store it into the `Name` variable and click a checkbox to store `TRUE` or `FALSE` into the `Gender` variable. The `DlgResult` variable contains the name of the button clicked to dismiss the dialog.



Dialog Presented with a Script Command

A custom dialog can be presented on any platform including Mac, Windows, Linux, iOS or Android. The user interface will be platform specific.

A subroutine of commands can be called before presenting a dialog to load its data. Another subroutine of commands can be called based on which button is clicked. In this abbreviated example, assume the dialog can be dismissed with two buttons **Cancel** and **OK**. If **OK** is clicked then run Sub\$OK else run Sub\$Cancel.

```
Button 1=Sub$Load|Dialog(...)|IfDoEvent(DlgResult,=,OK,Sub$OK,Sub$Cancel)
```

When testing a new dialog, use the **Variables** command on the **File** menu to present a window that shows the name and value of each variable after dismissing the dialog.

A Caption control may include only a Label or both a Label and Help string. This example adds a one-line description to the dialog.

```
Caption#This text explains how the dialog works
```

A Caption control that includes a Help string puts a blue ? at the top right of the dialog. When clicked by the user, a Help dialog is presented. The Help dialog may contain multiple long paragraphs of text where each is separated by a semicolon in the text. Do not include ", ", "#" or ")" in the help text.

```
Caption#My Description#Paragraph1;Paragraph2;Paragraph3;Paragraph4
```

A dialog may include multiple Caption controls, but at most one of them should have a Help string. If the Label field of a caption control is empty, there will be empty space in the dialog that could be used to visibly separate and group other controls.

Dialog Control Help

Edit, List and Switch controls can be assigned to a URL for presenting images, help information or videos. If help is available, a blue ? appears to the left of the control and when clicked will present the URL in the default web browser.

To assign a help URL to a control, add &url=someURL to the Default field of that control as illustrated here for an Edit control.

```
Edit#Name#NameVar#&url=http://whatsmyname.com
```

The optional Default field for a control can have a help URL, a default value, a default value plus a help URL or neither as illustrated here.

- &url=http://whatsmyname.com
- Tom
- Tom&url=http://whatsmyname.com

ReportBuilder

This command generates a consolidated report from a variable number of inputs. It can be used to generate a Material List, Purchase Order or Project Bid on a report sheet from inputs collected on other sheets.

```
ReportBuilder(ReportSheetRange,HeaderRefList,SegmentedSubHeader,QuantityIndex,  
UnitCostIndex,ExtendedCost,Condition,RoundUpQuantity,LastRow,InputTable1,InputTable  
2,...)
```

ReportSheetRange is of the form SheetName!A1:D50 and identifies the Maximum cell range into which a report can be generated including an optional Header row. All cells in the range are emptied before building the report.

HeaderRefList is a semi-colon separate list of sheet specific cell references of the form Sheet1!A1;Sheet1!B1;Sheet2!A1. This command gets a label from each referenced cell and uses it as a bold header for one column of the report. If HeaderRefList is empty, the report will have no header row.

SegmentedSubHeader field may either be empty or contain data to construct two types of reports.

If SegmentedSubHeader is empty, a consolidated report is generated where all rows with matching ItemName column from all InputTables are consolidated into one row in the generated report.

If SegmentedSubHeader contains data, a segmented report is generated. The SegmentedSubHeader field consists of cell references separated by semicolons with one cell for each InputTable field. For example assume your report is built from 40 tables and each table has a cell that identifies a section name like Section1, Section2 or Section3. The information for all tables assigned to Section1 gets consolidated into that part of the report, all tables related to Section2 gets consolidated into Section2 of the report, etc.

QuantityIndex refers to the item index of the Quantity column within the InputTable field where ItemName is at index 1.

UnitCostIndex refers to the item index within the UnitCost column within the InputTable field where ItemName is at index 1. If ExtendedCost is FALSE, the UnitCostIndex field can be 0 since it is not used.

ExtendedCost is TRUE or FALSE. If TRUE, then QuantityIndex and UnitCostIndex fields must be non-zero and an Extended Cost column is calculated and added to the right side of the constructed report.

Condition is a semi-colon, separated column range each having the same number of rows as items in the InputList. If the ReportBuilder references 20 InputTables, than this parameter contains 20 column ranges.

Each cell within the referenced column range contains TRUE or FALSE indicating if that row should be included in the report. If you don't use this feature the Condition field can be empty. Using conditional logic within the workbook or script you can flag which rows to include in the report. You could also allow the user to click checkboxes or have a user-clicked button run a script to determine which rows to include.

RoundUpQuantity is FALSE or TRUE to indicate if quantity values should be rounded up to a whole number after consolidation into a report.

LastRow is a Variable name that will contain the last used row number after the report is generated. This field can be empty if you don't need it. Knowing the last used row in the generated report will allow your script to construct a custom footer on the next row of the sheet. Your script already knows what columns are used by the report. The LastRow field is also useful when you want to use ReportBuilder command more than once on the same sheet since the report auto-sizes itself as needed.

InputTable is a semi-colon separated list of input column references from a table of input cells. For example, an input table containing columns ItemName, Quantity, Units, UnitCost might use an InputTable field of the form Sheet1!A1:A50;Sheet1!B1:B50;Sheet1!C1:B50;Sheet2!A1:A50. The first item in the list must refer to the ItemName that is used for consolidation in the generated report. If the ItemName matches on multiple rows, then Units and UnitCost should also match. In the consolidated report, the values for these rows come from the first consolidated row.

The input columns for a table are generally on one sheet. You can pull input data from columns on different sheets, but the number of rows in the range should be same for each item in the list. The ReportBuilder command may reference up to 100 different input tables, each referenced by a different InputTable field.

Columns are not resized when building a report so size them appropriately on the empty sheet before building the report. Make sure the output sheet has enough columns and rows to contain the ReportSheetRange field you specify in the command. If you want a big bold and centered title that spans multiple columns do that in the workbook itself or with script in the output sheet.

The ReportBuilder command is part of a script that generally runs after the user has entered input table data and clicks a button. That script could also show the output sheet that may have been previously hidden.

Cell Data

The `CellDataRead` and `CellDataWrite` commands are intended for experienced developers with a good understanding of how ExcelRT works internally. These commands can read or write properties of any cell on any sheet. The write command in particular should be used with great caution since invalid data can easily corrupt the ExcelRT file that leads to exceptions and other abnormal behavior.

```
CellDataRead(SheetID,Col,Row,PropertyName1=Var1,PropertyName2=Var2,...)
CellDataWrite(SheetID,Col,Row,PropertyName1=Value1,PropertyName2=Value2,...)
```

These commands identify a specific cell by the sheet ID, column number and row number. `SheetID` is an integer 1 to 15. For column A, use 1 in the `Col` parameter or for column C use 3.

Every cell in ExcelRT has a collection of properties that determine its value, the calculations performed, how data is formatted, its text font, size and color, etc. These properties are normally configured within Microsoft Excel prior to conversion to ExcelRT or maintained internally by ExcelRT.

To understand cell properties and valid values, present the Cell Data dialog for a selected cell in ExcelRT while in Design mode (with an XML file opened). Here is a list of cell property names used by the `CellDataRead` and `CellDataWrite` commands.

BackgroundColor, BarProperties, Borders, Choices, Comment, Custom, Dependency, Enabled, Format, Formula, LimitText, MergeArea, Pattern, TextBold, TextColor, TextFont, TextInvisible, TextItalic, TextJustify, TextJustifyV, TextMultiLine, TextSize, TextStrikeThrough, TextUnderline, TextWrap, CellType, Validation, Value

In this example, the background color is set to green and text color is set to red for cell C5 on the first sheet. Colors are represented as RGB values (Red.Green.Blue) where each component is a number between 0..255. The write command may include any number of property value pairs where the value portion can be a literal value as shown here or a named variable that contains the value.

```
CellDataWrite(1,3,5,BackgroundColor=0.255.0,TextColor=255.0.0)
```

In this example, the command returns a value of `true` or `false` into the variable named `BoldVar` depending on whether or not the text style is Bold.

```
CellDataRead(1,3,5,TextBold=BoldVar)
```

Cell Controls

Cell controls provide an advance feature that is only available with a script command. A script can create controls like checkboxes or radio buttons directly within a cell instead of defining them in the MS Excel file. The script can dynamically control the state or text of cell controls and even show or hide them by showing or hiding a column or row.

From Microsoft Excel, you can create two types of cells (Locked and Unlocked). In ExcelRT, locked cells show non-editable text and Unlocked cells provide an Edit box when selected. The Cell Properties dialog for a selected editable cell in ExcelRT, shows properties like Cell Type, Value and Choice.

The CellType property can hold several valid values:

- 0 = Non-Editable Text
- 1= Editable Text
- 2 = Popup Menu
- 3 = Combobox
- 4 = Checkboxes
- 5 = Picture
- 7 = Progress Bar
- 8 = Radio Buttons
- 10=Button

Non-Editable Text

The CellDataWrite command can be used to change the CellType. In this example, the cell at B3 on sheet 1 is changed to a non-editable cell.

```
CellDataWrite(1,2,3,CellType=0)
```

Checkbox

To create a checkbox named Adult in B3, use this command.

```
CellDataWrite(1,2,3,CellType=4,Choices=Adult)
```

If unchecked, the cell value is empty. When checked, the cell value contains the word Adult. In the following example, B3 contains three checkboxes and displays the result in a MsgBox when a button named ShowBtn is clicked.

```
CellDataWrite(1,2,3,CellType=4,Choices=Adult~Married~Citizen)  
ShowBtn=VarFromCell(State,B3)|MsgBox(State)
```

Notice how three values are supplied in the Choices property separated by the ~ character.

☒ Adult ☒ Married ☒ Citizen

ExcelRT actually expects a comma character to separate choice values, but since comma is used to separate command parameters in a script, you must use ~ instead.

If all three checkboxes are set, the MsgBox shows: `Adult,Married,Citizen`

Radio Buttons

In this example, two radio buttons are presented in cell B3 and Male is the default selection. When the user selects the Female radio button, the cell value is set to Female.



```
DefineVar(Sex,Male)|VarToCell(Sex,B3)|
CellDataWrite(1,2,3,CellType=8,Choices=Male~Female)
```

Popup Menu

In this example, when cell B3 is clicked a popup menu allows Red, Green or Blue to be selected. The cell now holds the selected value. When using a Popup menu control, you may first want to assign a default value to the cell. For iOS, user must double-click to present the Popup menu.

```
CellDataWrite(1,2,3,CellType=3,Choices=Red~Green~Blue)
```

ComboBox

To present a ComboBox instead of a Popup menu, use CellType 4. With a ComboBox, the user can pick one of the defined choices and type their own value.

For iOS, user must double-click to present the ComboBox menu.

Picture

A picture can be loaded into memory from an image file stored in the Plugins folder, then copied into a cell. The picture can be used like a button control to perform an action when clicked or to display user data. Pictures can be large and are not stored with the ExcelRT file itself.

Static pictures used in controls are generally supplied to the Plugins folder when the product is installed. If ExcelRT files are shared between devices, user provided pictures can be stored in the Plugins folder for offline use, but the master copy of the picture should be stored in the cloud for use across devices.

Progress Bar

To present a Progress Bar, use CellType 7 with the Min and Max value of the Progress Bar defined in the Choices property and the cell value represented by the bar itself.

```
CellDataWrite(1,2,3,CellType=7,Choices=1~100,TextColor=255.0.0)
```

Here is an example of the Progress Bar with a cell value of 25 when the Min and Max values are 1 and 100.



Button

Set the background color and center the text to create a rectangular button within a cell. Use OnCellClick[SheetID,Column,Row] event to run a script when clicked.

Sheet Resize

The total number of cells in a workbook will largely determine the file size and open and save time. For each sheet, add together the row count times the column count to determine the total cell count for a workbook.

By default, an ExcelRT file is typically sized to support the maximum columns and rows that will ever be required on each sheet. Some workbooks can be dramatically optimized to reduce file size and increase performance by adjusting the size of each sheet as needed to accommodate data or present temporary reports.

For example, a sheet could start with 10 columns and 10 rows or 100 cells. As data is entered, a script command can increase the sheet size to 20 rows, then 30 rows, etc. At 500 rows, the cell count is 5000 cells. The file will consume 50 times more disk space and take over 50 times longer to open.

Reports generated with the ReportBuilder command can be optimized for size and speed. Use the OnSheetActivate event to grow the sheet size and generate the report. Use the OnSheetDeactivate command to shrink it back to the minimize size. Use the OnClose event to return to the home sheet so all report sheets are minimized prior to saving the file.

Use this command to increase or shrink the columns and rows on a named sheet.

```
SetSheetSize(SheetName,ColCount,RowCount)
```

When a sheet is expanded, the new cells are completely empty. They contain no data or formatting. Perhaps you would like to left or right justify the text in those cells, give it a currency format or a text or background color. This command copies all the cell properties from a reference cell to an entire range of new cells.

```
SetCellStyle(CellRef,Range)
```


Feature Control

QuickLicense and Cloud License support Feature flags for a Serial Number activated Product within ExcelRT. Feature flags can be used to make sheets visible, show or hide a range of cells, enable buttons, scripts or whatever is needed within a workbook.

Feature flags are available to all instances of a workbook created from the Open Data File window. If a Serial Number allows activation on multiple devices, the same feature flags exist. If a license is released on one device and activated on another, the feature flags move with the license.

One hundred Feature flags are associated with each Serial Number and set with a checkbox from Safe Activation. Feature flags can be indirectly set from a shopping cart, InApp purchase, your website via the LicenseSupport API or in other ways.

When a product is activated within ExcelRT with a Serial Number, the workbook can access a 100-character string using the Features symbol. Each character is 0 or X where X indicates that feature is enabled. This command reads the Feature flag string into a variable named Flags.

```
SymbolValue(Flags,Features)
```

These commands extract character 1 into `IsF1`, then sets `S1Visible` to TRUE if it is an X and finally shows or hides Sheet9.

```
Mid(Flags,1,1,IsF1)|IfSetVar(IsF1,=,X,S9Visible)|SheetProperties(9,Sheet9,S9Visible)
```

When using QuickLicense for a desktop app, Feature flags are read from the Safe Activation server into the local device using the License Options dialog. On application launch, hold down the **Shift** and **Command** keys to present the License Options dialog, select the Refresh Features panel and click **OK**. Your ExcelRT script must first run command `QuickLicenseStandalone` to access the Features symbol.

```
QuickLicenseStandalone(Product,Security)
```

An InApp purchase within an ExcelRT product using Cloud License or QuickLicense will immediately set purchased Feature flags within the running Product.

Credit Card and Paypal

This section describes commands used to process credit card or Paypal payments from within a workbook. This process can only be used for an activated product. It requires a free Paypal merchant account to process credit card or Paypal payments and a Vendor account on Safe Activation Service 3 to register your Paypal merchant credentials.

Two payment processes are supported. Use either or both process to sell any number of items within your application. Purchased items may enable advanced features or sheets within your workbook or result in delivery of completely independent products or services.

- Vaulted Credit Card
- Paypal Payment

Only a couple commands are required to complete an InApp purchase as documented below. Additional information is available in an ExcelRT sample file, video or online help within a Safe Activation vendor account.

Alternatively, a **Purchase** button on the Open Data File window can be configured without any script commands. This approach is documented separately and will be the easiest and most convenient way to sell items for most projects.

These commands are only valid from within an activated product:

- Standalone Application
- Shared QuickLicense Product
- Shared Cloud License Product

Call this command from the `OnOpen` event or prior to other payment commands. The Product and Security fields must exactly match those fields in the main QuickLicense window for the generated Ticket.

```
QuickLicenseStandalone(Product,Security)
```

Vaulted Credit Card

This approach uses one command to vault credit card data in the Paypal payment processor and later a second command can be used to make a purchase on the vaulted credit card. The credit card is kept on file and can be used for multiple transactions. The user doesn't know that Paypal is processing the payment, they'll just see the product purchase from your company on their credit card bill.

This command vaults credit card data provided by the user. Result1 and Result2 are names for two return variables. Result1 will contain the values CANCEL, FAILED or SUCCESS. If SUCCESS, Result2 contains the CardID of the vaulted card.

```
PaypalVaultStore(Merchant,CardType,CardNumber,CardCVV,CardYear,CardMonth,CardFi  
rstName,CardLastName,Result1,Result2)
```

Merchant is an arbitrary string that you provide like YourCompanyName that identifies vaulted credit cards for that merchant within your Paypal account. CardType must be visa, mastercard, amex or discovery. CardNumber is the credit card number. CardCVV is a security code. CardYear is a 4-character string like 2018. CardMonth is a number 1 to 12. CardFirstName and CardLastName provide the name on the card.

This command prompts the user for card data and vaults the credit card data. It returns the same Result1 and Result2 variables.

```
PaypalVaultStorePrompt(Merchant,Result1,Result2)
```

After vaulting a credit card, call this command to store the CardID. That computer or device will remember the CardID so it can be used with future transactions.

```
SetCardID(MyCardID)
```

To retrieve a CardID, use the CardID symbol with the SymbolValue command.

```
SymbolValue(MyCard,CardID)
```

Use this command to make a purchase with a vaulted credit card.

```
PaypalVaultPay(CardID,ItemName,ItemQuantity,ItemPrice,ItemFeature,  
Currency,Description,Invoice,Shipping,Tax,Amount,Fields,Result1,Result2)
```

Parameters ItemName, ItemQuantity, ItemPrice, ItemFeature can represent one or more items for purchase. The ItemFeature is an integer value 1 to 100 to represent a Feature flag that is set if the purchase is approved or 0 if Feature flags are not used.

The Currency field is USD for US dollars. Description is a short text field that describes the purchase.

Every purchase must have a unique Invoice number or it will be rejected. The easiest to accomplish this is to leave that field empty (but present) and Safe Activation will provide a sequentially incrementing Invoice number regardless of which customer application makes a purchase.

Here is an example of the PaypalVaultPay command where quantity 2 of Item1 is used at a price of \$10.00 each. The total also includes \$5.00 shipping and \$1.25 tax. Keep in mind that this command is one line in the Script editor. The first parameter is the CardID returned by the original PaypalVaultStore command.

```
PaypalVaultPay(CARD-4WK07283PT214793KLHTCSAY_2018-04-30_1317,Item1,2,10.00,
,USD,Description,,5.00,1.25,26.25,email-excel@spinn.net,Result1,Result2)
```

Here is another example, where Item1 is associated with Feature flag 1 and Item2 is associated with Feature flag 2. Notice how ! is used to separate the names Item1 and Item2. Likewise, the quantity, price and feature flags fields are separated the same way.

```
PaypalVaultPay(CARD-4WK07283PT214793KLHTCSAY_2018-04-
30_1317,Item1!Item2,1!1,10.00!20.00,1!2,USD,Description,,5.00,1.25,26.25,
,Result1,Result2)
```

The Shipping and Tax fields can be empty or an amount. Make sure the Amount field reflects the total quantity times the price for each purchased item plus the shipping and tax. If Amount is incorrect, the order is rejected. If a payment is rejected, the Result1 variable has a value of FAILED or CANCEL. If FAILED, the Result2 value may provide the reason.

The Fields parameter can be empty or contain one or more field name and value pairs as illustrated here. Notice there are 3 field-value pairs separated by the ! character. The field names are company, phone and email.

```
company-Excel Software!phone-7024457645!email-excel@spinn.net
```

If payment is successful, Result1 value is SUCCESS and Result2 value is the assigned Invoice number.

The value of CardID has three parts separated by _ character. The first part is used by Paypal to look up the credit card data within their vault. The second part is the date that determines how long it is stored in the vault. This depends on the card expiration date and how long Paypal chooses to store card data. The final part is the last 4 digits of the credit card number.

```
CARD-4WK07283PT214793KLHTCSAY_2018-04-30_1317
```

Paypal Payment

This command presents a familiar Paypal checkout process presented within a dialog. The user navigates through a series of screens selecting the payment source and credentials within their Paypal account. Once the payment process is completed, the dialog will disappear.

```
PaypalAccountPay(Item1,1,10.00,1,USD,Description,,,,10.00,,Result1,Result2)
```

If a payment is rejected, the Result1 variable has a value of FAILED or CANCEL. If FAILED, the Result2 value may provide the reason. If the user clicks the Cancel button before the payment process is completed, then Result1 contains CANCEL and Result2 is empty.

If payment is successful, the Result1 variable is SUCCESS and Result2 variable is the assigned Invoice number. All other parameters work the same as they do in the `PaypalVaultPay` command.

Stripe Payment

Stripe payments can be processed within an ExcelRT application. To use this feature, the developer needs a Stripe and Safe Activation Service 3 account. From Safe Activation, configure a Product or Subscription button that presents the Stripe checkout page.

Typically a button on your website takes the user to the Stripe checkout page. After entering credit card data to complete the purchase, the user is returned to a completion page on your website where they can be given further instructions or download software. There are several ways to offer a similar experience within an ExcelRT application.

A script command can present a URL in the default web browser that handles the purchase process. In this example, when the user clicks Button1, a browser window is presented to complete the purchase. Alternatively, add a page to your website with a product description and the Buy button.

```
Button1=ShowURL(https://www.safeactivation.com/stripe.php?db=6&vendor=20080503&order=4&button=Product1)
```

The ShowHTML command presents a dialog that displays an HTML page. That page can be constructed within your script to include a description and a Buy Now button that presents the Stripe checkout page.

The Stripe checkout process can run within an HTML Form Control on a workbook sheet.

Form Control

Control Type: HTML

Name: Stripe

URL or Plugin: https://www.safeactivation.com/stripe.php?db=6&vendor=20080503&

Shape Top: 0

Left: 0

Width: 800

Height: 400

Top Left:

Row: 1

Column: 1

Top Offset: 0

Left Offset: 0

Bottom Right:

Row: 1

Column: 1

Bottom Offset: 0

Right Offset: 0

Placement: Free Floating

☒ Visible

Cancel
OK

In this example, the Form Control is presenting just a Buy button that when clicked presents the Stripe checkout process. Alternatively, an HTML source page in the Plugins folder could include a description, image and the Buy button.

Here is an example of the Stripe checkout process running within an HTML Form Control on a workbook sheet.

The screenshot shows a web browser window titled 'ExcelRT Builder:ODBC' displaying a Stripe checkout page. The page is divided into two main sections. The left section, titled 'Excel Software TEST MODE', shows a payment summary for 'Pay Excel Software' with a total due of \$2.95. The right section, titled 'Pay with card', contains a form for entering payment details. The form includes fields for Email, Card information (number, expiry, CVC), Name on card, Country or region (set to United States), and ZIP. A blue button labeled 'Pay \$2.95' is located at the bottom right of the form.

Message Dialog

These commands are used to present a message in a dialog.

This command presents a message dialog with one or more lines of text separated by commas. Only one parameter is required, but up to 10 can be included. Each line of text starts on a new line and is followed by a blank line. If the line is too long, it wraps to the next line. Adding comma at the end of the parameter list can increase the dialog height.

```
MsgBox(Text1,Test2*,...)
```

Display a movable message dialog with the supplied Title and specified Width and Height. Position can be TopLeft, Left, BottomLeft, TopCenter, Center, BottomCenter, TopRight, Right and BottomRight. There can be 1 to 10 lines of comma-separated text. Each chunk of text is displayed as a separate line although if long enough it wraps to the next line.

```
MsgBoxMove(Title,Width,Height,Position,text,text2*,text3*,...)
```

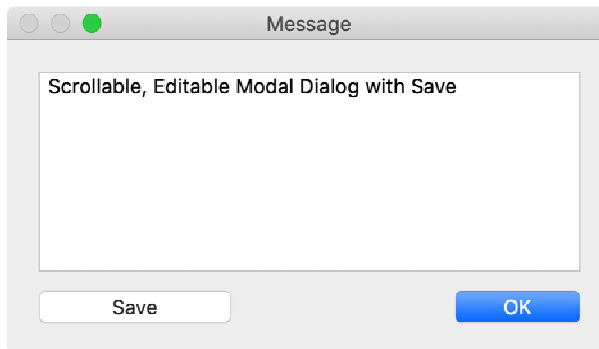
Display a movable modal dialog with the supplied Title and specified Width and Height. Position can be TopLeft, Left, BottomLeft, TopCenter, Center, BottomCenter, TopRight, Right and BottomRight. There can be 1 to 10 lines of comma-separated text. Each chunk of text is displayed as a separate line although if long enough it wraps to the next line.

```
MsgBoxModal(Title,Width,Height,Position,text,text2*,text3*,...)
```

Display a movable, resizable modal dialog with the supplied Title and specified Width and Height. Text displayed in the dialog is editable, scrollable and can be saved to the Desktop in a plain text file name specified by DlgTitle. Set Editable to TRUE to make displayed text editable by user. OK and Save provides the button name and if empty, makes that button invisible.

```
MsgBoxSave(Title,Width,Height,Position,Editable,OK,Save,text,text2*,text3*,...)
```

Position can be TopLeft, Left, BottomLeft, TopCenter, Center, BottomCenter, TopRight, Right and BottomRight. There can be 1 to 10 lines of comma-separated text. Each chunk of text is displayed as a separate line although if long enough it wraps to the next line.



In ExcelRT Cloud, dialogs are always centered and non-movable so MsgBoxMove and MsgBoxModal work the same and the Position parameter is always ignored.

Miscellaneous

This command is used to assign a format to a specified range of cells.

```
FormatCells(Range,Format)
```

Microsoft Excel uses the General format for cells that contain plain text. By default, ExcelRT can only read and write 1-byte encoded ASCII text in cell values. This works fine for most workbooks.

Some human languages cannot be represented with ASCII text characters. ExcelRT supports a special format named Unicode that is similar to General except that it does allow the cell value to read, write and display a Unicode string value.

During the conversion process, ConvertExcelRT can convert each cell with General format to Unicode format. Alternatively, if only a small subset of cells need to accept human entered Unicode text, use the FormatCell command in an OnOpen event to assign that Format to those specific cells as illustrated here.

```
OnOpen=FormatCells(Config!A5:D10,Unicode)
```

To recalculate the workbook after changing cells or importing data, use `Recalculate()`. The recalculation process is usually the most time consuming feature in a workbook and offers the best opportunity for speed optimizations.

The `Recalculate(Parm)` command accepts one parameter of TRUE, FALSE or a Sheet Title. A value of TRUE recalculates all sheets and FALSE only recalculates the current sheet. TRUE is assumed if no parameter is provided. To recalculate cells on a specific sheet, provide the sheet title as illustrated here:

```
Recalculate(Sheet1)
```

Use the `RecalculateRange(CellRange)` command to recalculate a specific cell range as illustrated below. The CellRange can be a literal value or variable name.

```
RecalculateRange(Sheet1!A1:G20)
```

The `SetRecalculate(Algorithm)` command can change the recalculation algorithm. The Algorithm parameter must be `RecalcFull`, `RecalcSheet`, `RecalcDependency` or `RecalcNone`.

To redraw the workbook, use `Redraw()`.

To save the workbook, use `Save()`.

The `Print()` command presents the Print Selection dialog, then prints the selected sheets.

To select a specific sheet, use `ShowSheet(SheetName)`.

To change the title or visibility of a sheet, use this command. ID is an integer value 1 to 15 or variable containing the value. Identify a sheet by ID number from left to right starting with 1. Title is a named variable or text representing the sheet name. The Visibility parameter is TRUE or FALSE or a variable containing that text.

```
SheetProperty(ID,Title,Visibility)
```

Do not change properties of the sheet that is currently displayed. The `SheetTitle` command is a bit unusual in that you can change the title of the current sheet. The screen is redrawn once the script completes. The title change is not persistent when you close the ExcelRT file on a desktop app or switch to a different screen on iOS.

```
SheetTitle(NewTitle)
```

Show and hide rows or columns with `RowVisible(SheetName, From, To, State)` and `ColumnVisible(SheetName, From, To, State)` as illustrated here.

```
RowVisible(Sheet1,2,3,TRUE)
ColumnVisible(Sheet2,3,3,FALSE)
```

These commands can get or set the row height and column width properties of one or more rows or columns. `From` and `To` is the start and end row or column number.

```
RowHeightSet(SheetName,From,To,Height)
RowHeightGet(SheetName,From,To,Height)
ColumnWidthSet(SheetName,From,To,Width)
ColumnWidthGet(SheetName,From,To,Width)
```

This command will stop the script and present the Variables window. View or edit variables, then close the window to continue the script processes. This command is ignored on iOS and Android.

```
Debug()
```

A line in the script may include comments. Add `//` at the beginning of the line to comment out the entire line. Add `//` somewhere after the `=` character to comment out the last part of a line of script. These examples, demonstrate both types of comments.

```
//Button 1=MsgBox(Hello1) |MsgBox(Hello2)
```

```
Button 1=MsgBox(Hello1)//MsgBox(Hello2)
```

```
// Here is a developer comment that is ignored when executing the script
```

This command changes the navigation order of data entry in a sheet when pressing the **Tab** or **Enter** key between entered values. By default, the selected cell moves left to right. Use a `State` parameter of `TRUE` to navigate top to bottom.

```
TabDown(SheetName,State)
```

These commands show or hide horizontal or vertical gridlines. The `State` parameter is `TRUE` or `FALSE`.

```
GridlineHorizontal(SheetName,State)
GridlineVertical(SheetName,State)
```

These commands show or hide the Header Row or Header Column for a sheet identified by name.

```
HeaderRow(SheetName,State)
HeaderColumn(SheetName,State)
```


This command presents the text contents of the Clipboard in a dialog and allows the developer to change the clipboard contents while executing the script. This command is useful for debugging purposes when reading and writing text data to the clipboard. For CSV formatted clipboard style, use Carriage Return (CR) to separate rows.

`ClipboardTextDialog()`

This command presents the text of a named variable into an editing dialog. The Title parameter can be text or a variable containing text to name the dialog.

`TextDialog(TextVar,Title)`

This command suspends script execution for specified number of milliseconds but allows other tasks like screen redraw to occur in a Desktop app. For ExcelRT Cloud, screen redraw only occurs when the script completes.

`DoEvents(MilliSeconds)`

Platform Specific Files

These commands only apply to Mac or Windows desktop applications. These commands read and write to platform specific file paths when interacting with a user or another program.

This command looks for the existence of a file at the specified path, then returns TRUE or FALSE in a named variable. FilePath is a literal string or variable name.

`FileExists(FilePath,VarName)`

This command deletes the file at the specified path. FilePath is a literal string or variable name.

`FileDelete(FilePath)`

These commands copy files to or from the Plugins folder. FileName identifies the file without a file path inside the Plugins folder. FilePath is a literal string or variable name containing a file path outside of the Plugins folder.

`FileCopyToPlugin(FilePath,FileName)`
`FileCopyFromPlugin(FilePath,FileName)`

This command prompts the user for a location on disk and then copies FileName from the Plugins folder to that location. FileExt is an optional parameter that allows you to force the file extension regardless of what the user enters. PromptText is an optional parameter that allows you to add description text to the dialog.

`FileSaveAsFromPlugin(FileName,FileExt*,PromptText*)`

In ExcelRT Cloud, the FileSaveAsFromPlugin presents a dialog with a download link. When clicked the file from the Plugins folder is downloaded to the Downloads folder of that computer. The Download command directly downloads the named file from the Plugins folder.

`Download(FileName)`

These commands prompt the user for a file or folder location. FilePath is the name of a variable that holds the full path to the selected file or folder. If the user cancels the presented dialog, then the FilePath variable is empty. InitialPath is an optional parameter that specifies the initial folder presented by the selection dialog.

```
PromptForFile(FilePath,InitialPath*)  
PromptForFolder(FilePath,InitialPath*)
```

This command returns a colon-separated list of files in folder. The optional Filter returns only files with a specific extension. Filter can be one or more semicolon separated file extensions of the form .jpg;.png.

```
FilesInFolder(FolderPath,ListVar,Filter*)
```

Run a command line (DOS or Unix shell command) and return the result if any in ResultVar. If the path to command contains spaces, it is safer to use optional Params command to provide parameters.

```
Shell(Command,ResultVar,Params*,ExitCode*,Pipe*)
```

ExitCode is an optional variable name that will contain 0 if success or a system supplied error code. Pipe is an optional parameter to provide a character replacement for a Linux Pipe since | is used to separate ExcelRT commands. For example, use ; instead of | in command, then use ; as the Pipe parameter.

Splitting & Scaling Images

When generating a large report from a sheet, it may not fit well on a printed page. A large image scaled down may become too small to read. On a desktop computer you may be able to split the print into multiple pages but some rows may get split between two pages.

This command generates one or more JPG files from the current sheet for the purpose of printing those images or sending them to another computer. This allows long reports with hundreds of rows to be readable when printed. It assumes that the width of all columns in the image is acceptable on the printed page.

```
SplitSheetToJPGs(BaseFileName,RowCount,PageCount,JPGFileList,Scale*)
```

Rows are split nicely on row boundaries with the maximum number of rows per image determined by RowCount. PageCount is a return variable to indicate the number of JPGs created. Generated JPG files are given a name from the BaseFileName and index number.

JPGListFile is a return variable name that stores a semicolon-separated list of generated image file names. This variable provides a convenient way to send those images to another user or computer by email or storage commands described in the Vendor Commands section.

Scale is an optional parameter that defaults to 2 if absent. A Scale value of 1 creates an image at screen resolution of 72 pixels per inch. A Scale value of 2 creates an image at 144 pixels per inch. The size of an image grows exponentially as Scale is increased so you'll seldom want a Scale value larger than 3 or 4.

Here is an example that creates three JPG images into the Plugins folder named Report1.jpg, Report2.jpg and Report3.jpg for a sheet containing 120 rows. The first image has 50 rows, the second has 50 rows and the third has 20 rows.

```
SplitSheetToJPGs(Report,50,PageCount,JPGFileList)
```

The SplitSheetToJPGs command generates images from the currently active sheet. The user must switch tabs to see each sheet.

To call the SplitSheetToJPGs command in a script that is initiated from a different sheet (like a button on Home sheet), you will need to use the following command to first navigate to the desired sheet and navigate back when done.

```
ShowSheet(SheetName)
```

This command is used to scale a JPG image or list of images. The image is scaled up proportionately until either the maximum width or height limit is reached.

```
ScalePluginJPG(Filename,MaxWidth,MaxHeight)
```

Filename is the name of a JPG file in the Plugins folder or a variable that contains the file name. You can supply a semi-colon separated list of file names. MaxWidth or MaxHeight is an integer or variable that contains an integer. By entering the desired pixel size of the image it can be adjusted to fit a piece of paper.

Here is an example:

```
ScalePluginJPG(Report1.jpg;Report2.jpg;Report3.jpg,800,1600)
```

This command is designed to create print-ready images by scaling them to fit on a piece of paper with desired margins. Paper and Margin sizes are given in inches. JPGList can be the name of an image file from the Plugins folder. It is typically a semicolon-separated list of file names when this command follows the SplitSheetToJPGs command.

```
JPGForPaper(JPGList,PaperWidth,PaperHeight,HorizontalMargin,VerticalMargin,Scale*)
```

Scale is an optional integer where 1 indicates 72dpi, 2 is 144 dpi, etc. If no parameter is supplied, the default scale is 2.

Plugin Files

Several commands are available to manage files in the Plugins folder. Use this command to rename a file in the Plugins folder. If NewFilename already exists, it is overwritten. The optional FolderName field can be the name of an existing folder within the Plugins folder or a nested folder using format Folder1:Folder2.

```
PluginFileRename(OldFilename,NewFilename,FolderName*)
```

Use this command to duplicate a file. If NewFilename already exists, it is overwritten. The optional OldFolderName and NewFolderName fields can be the name of an existing folder within the Plugins folder or a nested folder using format Folder1:Folder2.

```
PluginFileDuplicate(OldFilename,NewFilename,OldFolderName*,NewFolderName*)
```

Use this command to delete a file. If Filename doesn't exist, nothing happens. The optional FolderName field can be the name of an existing folder within the Plugins folder or a nested folder using format Folder1:Folder2.

```
PluginFileDelete(Filename,FolderName*)
```

Use this command to create a file in the Plugins folder from the supplied Variable or literal string. If Filename exist, it is overwritten.

```
PluginFileFromString(Filename,StrVar,FolderName*,Raw*)
```

Use this command to set the value of the named variable from the text contained within the named file in the Plugins folder.

```
PluginFileToString(Filename,StrVar,FolderName*,Raw*)
```

The optional FolderName field can be the name of an existing folder within the Plugins folder or a nested folder using format Folder1:Folder2

When reading or writing a plugin file from a string, use # to represent a carriage return and linefeed within the string or ~ to represent a comma. For example, this command creates a text file named MyFile containing 3 lines as illustrated below.

```
PluginFileFromString(MyFile,Line1#Line2#Line3 1~2~3)
```

MyFile

Line1

Line2

Line3 1,2,3

The optional Raw parameter can be set to TRUE to override the substitution behavior. For example, assume you want to set a string variable from data in a text file, then load a CSV memory structure with data from that string. You need to retain CRLF separating rows and , characters separating commas. Set the Raw parameter to TRUE.

To check for the existence of a file in Plugins folder, use this command that specifies the FileName and the name of a variable into which is stored TRUE or FALSE. The optional FolderName field can be the name of an existing folder within the Plugins folder or a nested folder using format Folder1:Folder2.

```
PluginFileExists(FileName,VarName,Folder*)
```

This command clones the current ExcelRT file into the Plugins folder. If the BaseFileName parameter is empty, the new file has the same name. If not empty, the new file is named with the BaseFileName plus the applicable .xml or .ert extension.

```
CloneToPlugin(BaseFileName)
```

This command could be combined with another command to email, upload or store the file online for use by another computer or person.

This command creates a folder in the Plugin folder if it does not already exist. To create a nested folder, FolderName can use form Folder1:Folder2.

```
PluginFolderCreate(FolderName)
```

This command deletes a folder in the Plugin folder if it exists. To delete a lower nested folder, FolderName can use form Folder1:Folder2.

```
PluginFolderDelete(FolderName)
```

This command checks for the existence of a folder in the Plugins folder. Specify the FolderName and the name of a variable into which is stored TRUE or FALSE. For a nested folder, FolderName can use form Folder1:Folder2.

```
PluginFolderExists(FolderName,VarName)
```

This command presents a Text Editor to create or edit plain text files in the Plugins folder. If Filename is provided, the text file is opened. FolderName can reference a folder in the Plugins folder or a nested folder using Folder1:Folder2.

```
PluginTextEditor(Filename*,FolderName*,Title*,BtnNames*)
```

A non-empty Title string replaces the dialog title. BtnNames is a semicolon-separated list of button names to rename or hide buttons. If the name is empty in the semicolon-separated list, that button is not displayed. Also include the word Radios as the last item in list to show the Line Ending radio buttons in the Text Editor dialog.

```
PluginTextEditor(MyData.txt,,,New;Open;Save;Find;Done;Radios*)
```

This command retrieves an image from the Internet and stores it in a file in the Plugins folder.

```
PluginFileFromURL(Logo.jpg,LogoUrl)
```

This command replaces the image displayed by a Picture control.

```
ReplacePictureFromPlugin(TitleSheetName,LogoPicture,Logo.jpg)
```

Google Map and Direction

These commands can be used to present a specified location on a Google generated map or the directions between two specified locations.

The `GoogleMap` command presents a map. It must include at least two parameters. If additional parameters are supplied, intermediate parameters are required even if empty. Each parameter can be a literal parameter with or without quotes or a variable name.

```
GoogleMap(Title,Location,Address,City,State,Country,Zip,Size,Position)
```

Here is a simple example, the presents a map showing the Empire State Building.

```
GoogleMap(Landmark,Empire State Building)
```

If the Title parameter is `Browser`, the google map is presented in the default web browser independent of the ExcelRT application. The Size and Position parameters are ignored.

If the Title parameter is not `Browser`, that text is presented in the title of a browser dialog. The Location parameter supplies a landmark location like "Empire State Building" or can be empty. Not all parameters are required. If you supply the City and State, Google can guess the Zip or vice versa.

The Size parameter can be empty, `Maximize` or `WxH` where W is a pixel width and H is a pixel height. The Position parameter can be empty, `TopLeft`, `Center`, `TxL` or `RelativeTxL` where T represents Top position in pixels and L represents Left position in pixels.

The `RelativeTxL` is a relative top and left position from the top left corner of the ExcelRT window itself rather than the top left corner of the screen. Here is a valid example where the presented Browser window is offset down 100 and left 200 pixels from the top left corner of the ExcelRT window.

```
Relative100x200
```

This example shows a map containing the Empire State Building in a window 1200x800 pixels positions centered on the screen.

```
GoogleMap(New York Landmark,Empire State Building, , , , ,1200x800,Centered)
```

The GoogleDirection command presents a map with directions and a path drawn between two locations on the map.

```
GoogleDirection(Title,FromLocation,FromAddress,FromCity,FromState,FromCountry,FromZip,ToLocation,ToAddress,ToCity,ToState,ToCountry,ToZip,TravelMode,Size,Position)
```

If the Title parameter is Browser, then google direction is presented in the default web browser independent of the ExcelRT application. The Size and Position parameters are ignored. If the Title parameter is not Browser, that text is presented in the Title of the presented browser dialog.

This command requires a minimum of 8 parameters and the remaining are not required. Required parameters can be empty as illustrated in the GoogleMap command. The parameters are similar to the GoogleMap parameter except a From and To location is specified. The TravelMode parameter can be empty or one of these specific values `driving`, `walking`, `bicycling` or `transit`.

Pictures

These commands are used to store a picture in a cell. When an ExcelRT file is opened, the named picture list is empty. A command can load a picture from an image file in the Plugin folder, manipulate the picture and assign it to a cell. A cell picture can present user data or represent a control that can be clicked to run a script.

A Picture can be referenced by its name and used while the file is open, but is not saved with the ExcelRT file itself. To make ExcelRT files portable across devices, use script commands to store the master copy of each picture in the Cloud and keep a local copy in the Plugin folder.

Use these commands to load or save a picture from an image file in the Plugin folder. If Scale is 1, no scaling occurs. Scaling can be useful when using the same picture across multiple platforms by assigning a platform specific scaling variable.

```
PluginToPicture(FileName,PictureName,Scale)
PluginFromPicture(FileName,PictureName)
```

Assign a picture to a cell using the PictureName.

```
CellPicture(SheetName,Col,Row,PictureName)
```


This command can be handy to extract dozen or hundreds of control images from a single image.

```
PictureCopy(PictureName,X,Y,Width,Height,NewPicture)
```

This command returns the Width and Height of a picture into the named variables.

```
PictureSize(PictureName,Width,Height)
```

This command will scale a picture to NewWidth and NewHeight, thus changing the original picture.

```
PictureScale(PictureName,NewWidth,NewHeight)
```

This command will copy FrontPicture onto a BackPicture to create NewPicture. For example, it could be used to add a logo in the top left corner of a user-supplied picture.

```
PictureToPicture(BackPicture,FrontPicture,X,Y,NewPicture)
```

This command presents the named picture in a titled dialog. It is ignored on iOS.

```
PictureShow(Title,Picture)
```

This command only exists on iOS and is ignored on other platforms. It can be used to take or access pictures from the app. Source indicates where the picture is coming from and should be one of these words, Camera, CameraRoll or PhotoLibrary.

```
PicturePick(Source,PictureName,Result)
```

Result is the name of a variable that will be SUCCESS or FAILED. If SUCCESS, the picture is stored in memory and assigned PictureName. A script can then present the picture in a cell or save it to a file in the Plugins folder.

This command creates a named picture in memory from a picture used in a form control. A fixed PictureID is used to reference a form control picture. Once the picture is named, it can be used in a cell control.

```
PictureName(PictureID,Picture)
```

Form Controls

These commands are used to manipulate Form Controls from a script. Form controls mimic those added to workbook sheets in Microsoft Excel. Use this command to make a form control visible or invisible.

`FormControlVisible(Name,State)`

The Name field is used to identify the control. The State field can be TRUE or FALSE or a variable holding that value. Form control data is loaded into screen controls when an ExcelRT file is opened. When you change the visibility of a control, it is not instantly reflected on the screen.

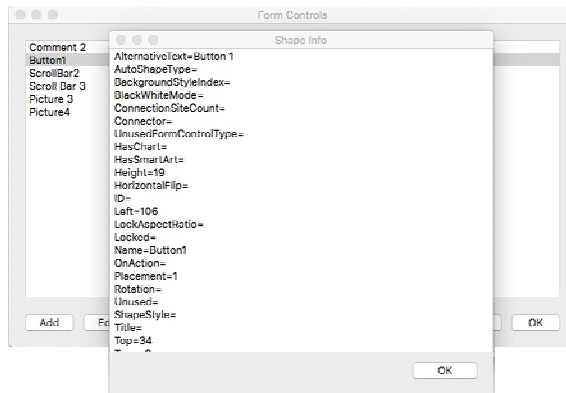
To rebuild all screen controls to reflect changes made with a `FormControlVisible` or `FormControlSet` command, use this command.

`FormControlRefresh()`

This command provides a low level means of poking in raw data from which Form Controls are constructed. The Name field identifies the control. The ItemIndex field is an integer starting with 1 the data string you want to change. The Value field provides a string for that data. These parameters can be literals or variable names that contain the actual data.

`FormControlSet(Name,ItemIndex,Value)`

To use this command, a programmer must determine the ItemIndex. Learning how to accomplish that is best illustrated with an example. Using ExcelRT Builder, double-click on the Form Control Edit tool, then select a control and click the **Info** button.



The Shape Info dialog shows the raw data used to build a Form Control on the screen. In simple terms, each line is a different ItemIndex. In this example, AlternativeText is index 1 and Left is at index 13.

The format of raw shape data is not fully documented anywhere. ExcelRT does not use some of these fields. This data comes from a converted Excel workbook so Microsoft's Excel VBA documentation may be helpful.

The AlternativeText field determines a button's displayed name. Assuming the Name field of the button is Button1, change the name on the screen to MyButton using these commands.

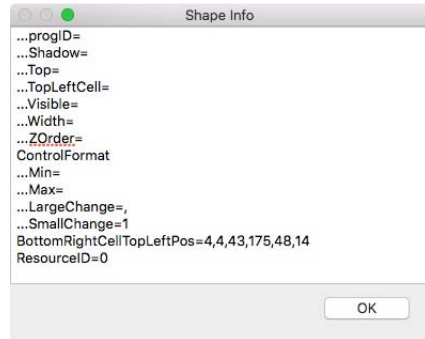
```
FormControlSet(Button1,1,MyButton)
FormControlRefresh()
```

Some fields in the raw data are more complex. For example, the BottomRightCellTopLeftPos field consists of 6 comma-separated parts.

Since commas are used to separate parameters in an ExcelRT Script command, you cannot easily supply this value. Instead, use ~ in place of each comma.

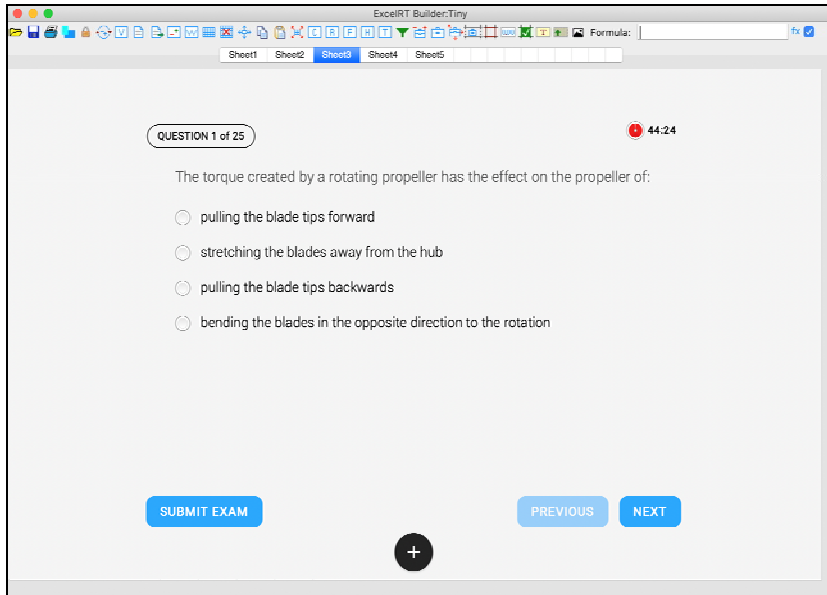
The ControlFormat field consists of four sub fields named Min, Max, LargeChange and SmallChange. Although the FormControlSet command cannot be used to change the value of this raw control data, remember to count this as just 1 item index when changing fields below it.

The FormControlSet command is not intended for a novice ExcelRT developer since one wrong command can corrupt an ExcelRT file making it unreadable. Keep current backups of your work when using this command.



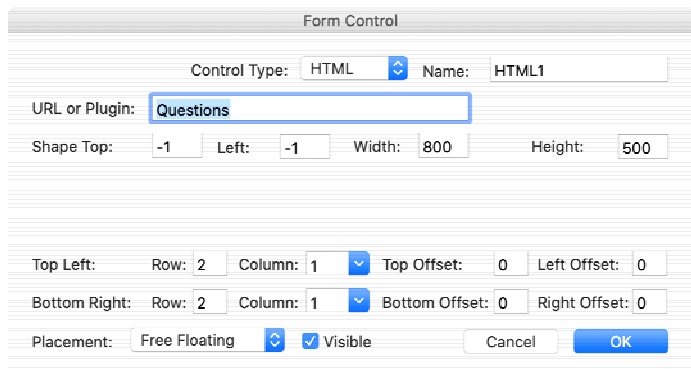
HTML Control

An HTML Control is a type of form control that can be added to a sheet. It can present an HTML file or collection of files from the Plugin folder or from the Internet. Use it to present interactive or online content in your workbook using Web technologies like HTML, CSS, Javascript or Java.

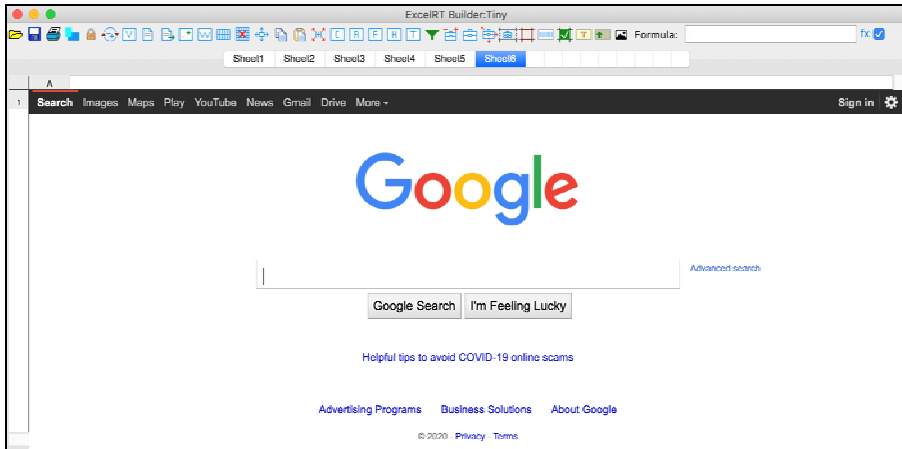


To display HTML based content from the Plugins folder, add a folder holding your HTML files. The main file must be named index.html.

In the Form Control dialog, notice the Control Type is HTML and the URL or Plugin field is the name of that folder within the Plugins folder.



This example shows the live Google website displayed on a sheet of your workbook. Enter a URL starting with http into the URL or Plugin field of the Form Control dialog.



Up to 5 HTML controls can be added to each sheet of a workbook. Position and size the HTML control as needed at design time using ExcelRT Builder.

Use the `FormControlVisible` command to show or hide HTML controls from a script. The `FormControlSet` command can even be used to change the URL or Folder name that determines the displayed content.

For example, a **Play Video** button could be added to your workbook. When clicked an HTML control is made visible on the sheet and a video from your website starts playing in the app. The button could be renamed to **Stop Video** to hide the video.

ExcelRT Cloud supports the HTML control with content from the Internet using a URL. It also supports content from the Plugins folder.

See ExcelRT Plugin section in the ExcelRT Builder chapter.

These commands are related to HTML controls and compressing or expanding plugins (typically for an HTML control).

This command compresses `FolderName` in the Plugins folder to `FolderName.excelrt_plugin`.

```
PluginCompress(FolderName)
```

This command expands `FolderName.excelrt_plugin` to `FolderName` in the Plugins folder.

```
PluginExpand(FolderName)
```

This command generates a file from a tagged template in the Plugins folder. This command is typically used to generate an `index.html` file containing data from workbook cells that can then be displayed with HTML, CSS or Javascript and presented within an HTML control.

```
PluginTaggedTemplate(Template,Filename,Array,FolderName*)
```

Template is the template file name, while Filename is the generated file created by replacing tagged values with data from the array. Array can be a single Array name or multiple array names separated by semicolons.

FolderName can reference a folder in the Plugins folder or a nested folder using Folder1:Folder2. If a password-protected plugin is used that is never expanded within the Plugins folder, FolderName can refer to the Plugin name such as Barchart.excelrt_plugin.

This command retrieves data from an HTML element named ExcelRTcopy within the HTML content displayed in an HTML control. Array is the name of an array in ExcelRT that holds the data. Status is a variable that returns TRUE or FALSE. Status is FALSE if the named array does not exist in a visible HTML control on the current sheet.

HtmlViewerCopy(Array,Status)

After retrieving array data, the value of the HTML control is clear. Refer to the HTML Control section in the ExcelRT Builder chapter for an example.

This command pastes text to the HTML content displayed within the named HTML control. Text is assigned to the value of an element named HTMLpaste within the HTML content. ControlName is the name of the HTML control in ExcelRT. Data is literal text or the name of a variable that holds the text. Status is a variable that returns TRUE or FALSE. Status is FALSE if the named HTML control does not exist on the current sheet.

HtmlViewerPaste(ControlName,Data,Status)

The HtmlViewerCopy and HtmlViewerPaste commands are not supported in ExcelRT Cloud and are simply ignored. Refer to the HTML Control section in the ExcelRT Builder chapter for an example. A compressed, protected or shared ExcelRT Plugin file can be used by an HTML control. Refer to the ExcelRT Plugin section in the ExcelRT Builder chapter.

This command redraws HTML controls on the screen.

HtmlRedraw()

This command constructs an HTML page from data in the current CSV and stores it in a named variable HtmlVar. If you use multiple CVS, use the CsvActive command to set the current Csv before calling this command.

CsvToHTML(HtmlVar,Headers*,BoldFirstRow*,Margin*,Grid*,Align*,BGColor*)

Header is an optional parameter where TRUE adds a column and row header. BoldFirstRow is an optional parameter where TRUE bolds text in first row. Margin is optional integer that adds space to cells. Grid is optional parameter where TRUE shows grid lines. Align is optional parameter of left, center or right. BGColor is optional background color of form rrggbb such as F2FBFF.

Command Log

These commands are used to debug and optimize command scripts. When logging is turned on, each processed command is assigned a timestamp and logged to the `Commands.log` text file in the `Plugins` folder.

This command turns on logging and creates an empty `Command.log` file when the `State` parameter is `TRUE`. If `State` is `FALSE`, logging is turned off, but the `Command.Log` file is not deleted.

```
CommandLog(State,VarList*)
```

`VarList` is an optional parameter that can be one or more variable names separated by semicolons. Those variable names and values are included on the log line.

Sometimes you want to add markers to the log file so you can keep track of where you are in the script. Logging must be on to use this command. This is also a handy way to output variable names and values at just that spot in the script.

```
CommandMark(Marker,VarList*)
```

Here is an example:

```
CommandMark(Look Here,Name1;Name2)
```

Sometimes you want to stop the script and present the `Command.log` file in a dialog. When you dismiss the dialog, the script keeps running. You can of course view the `Command.log` file in the `Plugins` folder from a plain text editor, but that is not convenient to do from iOS.

```
CommandShow()
```

Arrays

An ExcelRT workbook may create, resize, access and delete arrays as needed. There are no limits to how many arrays can be used by the workbook but only 10 arrays may exist at any one time.

Each array has a name, size and may contain any number of elements. An array element is accessed with the notation `ArrayName[Index]` where `Index` is a literal integer like 1, 2 or 3 or a variable name that holds the index. The first element of an array is at index 1 and the last element is at an index given by the array size.

An array reference can be used for any ExcelRT command parameter where a variable name is accepted. Array commands can simplify and reduce the number of commands in your script. More importantly, they are exponentially faster than loops they typically replace.

Before you can use an array you must create it. This command creates an array named `ArrayName` with `ArraySize` elements that are given a `Default` initial value. `Default` is a literal string or `Variable` name holding a value. If that named array already exists, it is sized smaller or bigger. If sized bigger, then existing values are retained and new value are given the `Default` value.

```
ArrayDefine(ArrayName,ArraySize,Default)
```

This command deletes the named array.

```
ArrayDelete(ArrayName)
```

This command returns the array size into the variable named `ResultVar`.

```
ArraySize(ArrayName,ResultVar)
```

This command assigns the `Value` (literal or `Variable` value) to the array element at `Index` (literal or `Variable` value) in `ArrayName`. If `ArrayName` does not exist or `Index` is less than 1 or greater than its `Size` then ignore command.

```
ArraySet(ArrayName,Index,Value)
```

This command copies the indexed array value into `ResultVar`.

```
ArrayGet(ArrayName,Index,ResultVar)
```


This command creates or resizes a named array. The size is based on the number of cells in CellRange, the value of the Option parameter and the data in each cell. Data from the referenced workbook cells is copied into the array elements.

CellRange examples include A1:A100 or SheetX!B1:C50 both of which create a 100 element array. The Option parameter must be ALL or NOEMPTY. If ALL option is used, then array elements are created from empty cells.

`ArrayLoad(ArrayName,CellRange,Option)`

This command will store array values into CellRange. If array size is smaller then cell count, some cells are not copied to. If array size is larger than cell count, some array values are not saved to the workbook.

`ArraySave(ArrayName,CellRange)`

This command will call the named Sub for each element of the array. IndexVar is the name of a variable that contains the index value of that loop iteration. ValueVar is the name of a variable that contains the array element value.

`WhileArray(ArrayName,IndexVar,ValueVar,Sub)`

ExcelIRT has a collection of commands to manage a delimited list of strings. Some features are easier to implement with a list and some with an array. This command will create a named list from the elements of an array. This is similar to a Join command in other programming languages.

`ArrayToList(ArrayName,ListName,Delimiter)`

This command will create an array from a named list. This is similar to an Explode command in other programming languages.

`ArrayFromList(ArrayName,ListName,Delimiter)`

This command will search an array for an exact matching Value and return the 0 or the found index in IndexVar.

`ArrayFind(ArrayName,Value,IndexVar)`

This command will add a new array element containing Value.

`ArrayAdd(ArrayName,Value)`

Cloud License

These commands are primarily used to access the Cloud License system within a Standalone iOS or Android app. If your iOS and Android solution runs within the ExcelRT host application or you are building a desktop application, you can probably ignore this section.

Assume you are building a Standalone App for iOS or Android. When a Standalone App is launched from a custom icon on the device, it opens directly into the Open Data File screen. There is no separate download or activation process required.

Within your ExcelRT workbook, you may want to enable additional features using a Serial Activation process configured with CloudLicense. The commands discussed in this section show how to accomplish that.

Using AddLicense, present the Cloud License dialog to configure a license that matches with a license configured in your Safe Activation account. Set the iOS and Android checkboxes to indicate which platforms the license applies to. Set the No License checkbox to indicate that the license is not applied within the Product view when testing your file in ExcelRT. The Run file for your App contains this data.

Use this command to activate or validate the license. This command is usually assigned to a button in your ExcelRT file. The first time a user clicks it, an Activation dialog is presented where the user can enter a purchased Serial Number.

```
CloudLicense()
```

Use this command to present a License Options screen where the user can view their Serial Number, release a license or perform other license maintenance activities.

```
CloudLicenseOptions()
```

Your ExcelRT file can determine if a license has been activated using the CloudLicenseStatus symbol, then show invisible sheets or take other actions. In the same script below, the license status is displayed when the ExcelRT file is opened.

```
OnOpen=SymbolValue(Status,CloudLicenseStatus)}MsgBox(Status)
```

Once a license has been activated, other symbols can provide valuable information about your activated license.

```
RequestNumber, SerialNumber, VendorID, Features, ProductID
```

Progress

These commands are used to show a progress screen during long scripting operations. At a minimum, use `ProgressShow` before starting the long operation and `ProgressHide` after it is completed to remove the progress screen.

Use this command to present a progress screen containing a title and message. The `ShowBar` and `ShowWheel` parameters can be `TRUE` or `FALSE` to include an optional progress bar and progress wheel. The optional `Msg2` parameter adds additional text at the bottom of the screen.

```
ProgressShow(Title,Msg1,ShowBar*,ShowWheel*,Msg2*)
```

Use this command repeatedly while the progress screen is active to update the text and progress bar value. The `BarValue` parameter is a value from 0 to 100.

```
ProgressUpdate(Msg1,BarValue*,Msg2*)
```

Use this command to hide the progress screen.

```
ProcessHide()
```

This command shows or hides a tiny spinning progress wheel at the top right corner of the ExcelRT window. The `State` parameters can be `TRUE` or `FALSE`.

```
ProgressWheel(State)
```

This command shows or hides a short progress label at the top right corner of the window. The `State` parameters can be `TRUE` or `FALSE`. The `Title` may be a named variable or literal text like 29 % or 1/9.

```
ProgressLabel(State,Title)
```

Switch Rows and Columns

These commands are used to switch rows and columns or a range of cells within those rows or columns. This allows a script to insert rows, remove rows or move cell data within a portion of the row up or down or the column left or right.

All cell properties are moved, not just the cell value. That includes color, font, formulas, etc. These commands are often used together with the `SelectedRow` and `SelectColumn` symbols. When a user clicks a button to trigger your script, your script can determine the currently selected row and column and act accordingly.

```
SwitchRows(SheetName,FromRow,ToRow,Redraw)
```

```
SwitchRowsRange(SheetName,FromRow,ToRow,FromCol,ToCol,Redraw)
```

```
SwitchColumns(SheetName,FromCol,ToCol,Redraw)
```

```
SwitchColumnsRange(SheetName,FromCol,ToCol,FromRow,ToRow,Redraw)
```

The `Redraw` parameter is `TRUE` or `FALSE` and indicates if the screen should be redrawn. When calling these commands in a loop, you might want to wait until the loop is finished then use a `Redraw()` command to optimize the speed.

Use this command to change the selected column or row:

```
SelectCell(Col,Row)
```

Sound

To create an audio beep, use:

```
Beep()
```

To speak, use:

```
Speak>Hello World)
```

This command plays an audio file from the Plugins folder. Folder is an optional folder name within the Plugins folder. Over a dozen file formats are supported depend on the OS. Formats include .m4a, .mp3, .wav, .aac, .ac3, .aiff and .flac.

```
PluginAudioPlay(FileName,Folder*)
```

This command stops an audio file if currently playing. This command is not currently supported on ExcelRT Cloud.

```
PluginAudioStop()
```

This command sets the volume level of an audio file from 0 (mute) to 100 (normal). This command is not currently supported on ExcelRT Cloud.

```
PluginAudioVolume(Level)
```

This command plays an instrument with specified Pitch and Velocity for the specified number of milliseconds. Instrument is an integer from 1 to 128. Use 0 for the Instrument parameter to present a dialog showing all instrument names.

```
NotePlayer(Instrument,Pitch,Velocity,MilliSeconds)
```

Pitch is integer 0 to 127 where middle C is 60. Incrementing by 1 raises pitch by a half step. Velocity is an integer up to 127 representing a very hard key press.

This command is supported on Mac computers using QuickTime and on Windows using the MIDI functions. The NotePlayer command is ignored on ExcelRT Cloud.

Python

Python is a programming language used by millions of developers. Python can be installed on macOS, Windows or Linux computers. All macOS computers have Python preinstalled.

This command will run a Python script stored in the Plugins folder giving it the Input parameters and storing the results in the Output parameter.

```
Python(Script,Input,Output,Timeout*,Executable*,FolderPath*)
```

The optional Timeout parameter in seconds will limit the amount of time ExcelRT will wait for completion, otherwise it defaults to 1 second. Executable is an optional symbol to launch Python that is almost always left empty. When empty it defaults to python unless python 3 has been installed on Mac when it defaults to python3. FolderPath is an optional folder path if the Script file does not reside in the Plugins folder. Here is an example that runs script.py.

```
Python(script.py,InputVar,OutputVar)
```

This command is supported in a macOS or Windows desktop application and may require the user to install Python on their computer. To use Python from any desktop application or ExcelRT Cloud with no user requirements to install Python, see the PythonServer command in ExcelRT Vendor Commands.

JSON

JSON is a lightweight data interchange format. It is often used to communicate structured data to an Internet source. It also provides a convenient way to collect data queried from a database or stored in a formatted text string.

The formatting rules for JSON are simple with good online tutorials. Each piece of data is stored in a key/value pair as represented here. Notice the first key is FirstName with a value of John. The second key is LastName with a value of Doe. Each key and value is enclosed within double quotes and separated with a colon. Multiple key/value pairs are separated with a comma and nested within curly braces that designate an object.

```
{"FirstName": "John", "LastName": "Doe"}
```

Another concept is an array where multiple objects are nested within brackets and separated with commas.

```
[{"FirstName": "John", "LastName": "Doe"}, {"FirstName": "Jane", "LastName": "Smith"}]
```

In this example, John Doe is in the first array element and Jane Smith is in the second element.

Script commands can be used to read or write a JSON object from a string or get and set data within that JSON object. This command creates a JSON object from a string formatted with JSON data.

```
JsonFromString(StrVar)
```

This command stores a JSON object into a string contained in a named variable.

```
JsonToString(StrVar)
```

These commands load a JSON object from a string, then save that object to a different string called Json2.

```
DefineVar(Json1, {"FirstName": "John", "LastName": "Doe"})  
JsonFromString(Json1)  
JsonToString(Json2)
```

This command extracts a value from JSON object based on the Indexer parameter. ResultVar is a variable name that contains SUCCESS or FAILED. Indexer consists of a string of integer indexes or names separated by periods.

```
JsonGetValue(Indexer, ValueVar, ResultVar)
```

For example, the first script command retrieves the name John into a variable named FirstName and the second command retrieves Doe into a variable named LastName.

```
JsonGetValue(FirstName, FirstName, ResultVar)  
JsonGetValue(LastName, LastName, ResultVar)
```

This command puts a value into a JSON object based on the Indexer parameter. ResultVar is a variable name that contains SUCCESS or FAILED. Indexer consists of a string of integer indexes or names separated by periods.

```
JsonSetValue(Indexer, ValueVar, ResultVar)
```

For example, assume the JSON object contains the name John Doe. This command will change it to John Smith.

```
JsonSetValue(LastName, Smith, ResultVar)
```

Assume you have loaded a JSON object with an array of first and last name pairs. Use the indexer parameter to specify the second element of the array and retrieve the FirstName field. With the correct indexer string, retrieve any data value from any JSON object.

```
JsonGetValue(2.FirstName, FirstName, ResultVar)
```

This command extracts an array of values from JSON object based on the Indexer parameter. ResultVar is a variable name that contains SUCCESS or FAILED. Indexer consists of a string of integer indexes or names separated by periods.

```
JsonGetArray(Indexer, ArrayVar, ResultVar)
```

Assume a JSON object contains an array where each element is an object with FirstName and LastName key/value pairs. This command returns the FirstName of each object in the array into MyArray. The first part of the indexer string identifies the array within the JSON object, while that last part identifies the key of the object from which to retrieve a value.

```
JsonGetArray(FirstName, MyArray, ResultVar)
```

This command puts an array of values into a JSON object based on the Indexer parameter. ResultVar is a variable name that contains SUCCESS or FAILED. Indexer consists of a string of integer indexes or names separated by periods.

```
JsonSetArray(Indexer, ArrayVar, ResultVar)
```

The indexer parameter for the `JsonGetArray` and `JsonSetArray` commands can access a range of objects within the JSON object. For example, assume the JSON object looks like this.

```
{{"F": "John", "L": "Doe"}, {"F": "Jane", "L": "Doe"}, {"F": "Sue", "L": "Smith"}, {"F": "Cindy", "L": "Gray"}}
```

Assume we want to get the first name Jane and Sue into a two element array. This special indexer would do it.

```
JsonGetArray(F.from2to3, MyArray, ResultVar)
```

The F part of the indexer string indicates you want the first name from each of object of the array. The `fromXtoY` construct indicates that you want the Xth object to the Y object from the array starting with an index of 1.

Assume the JSON object has many nested arrays and objects. This indexer indicates an array of Duck objects at the `Animal.Bird.Duck` level. Perhaps the array contains many characteristics of 100 different species of Ducks. From this array, we want to access objects 5 through 9 and copy the Species name into a 5-element array.

```
Animal.Bird.Duck.Species.from5..9
```

An indexer is almost always used to retrieve values from key/value pairs. In special cases, you might want to retrieve the key name itself. In this example, the key names F and L are retrieved from the first element of the array.

```
JsonGetArray(keys, MyKeys, ResultVar)
```

Assume the JSON object has this data. Note the data is not an array, but instead a single object with multiple key/value pairs.

```
{ "First": "Tim", "Last": "Gray", "Age": "27", "Weight": "160", "Job": "Carpenter", "State": "NY" }
```

Use this command to retrieve an array containing First, Last, Age, Weight, Job and State.

```
JsonGetArray(keys, MyKeys, ResultVar)
```

This revised indexer returns the key names First, Last and Age into an array named `MyKeys`.

```
JsonGetArray(keys1to3, MyKeys, ResultVar)
```

Assume you want to retrieve the values Tim, Gray, 27... from the Json object. Use this indexer convention.

```
JsonGetArray(node, MyKeys, ResultVar)
```

To retrieve only the first and last name, use this indexer value:

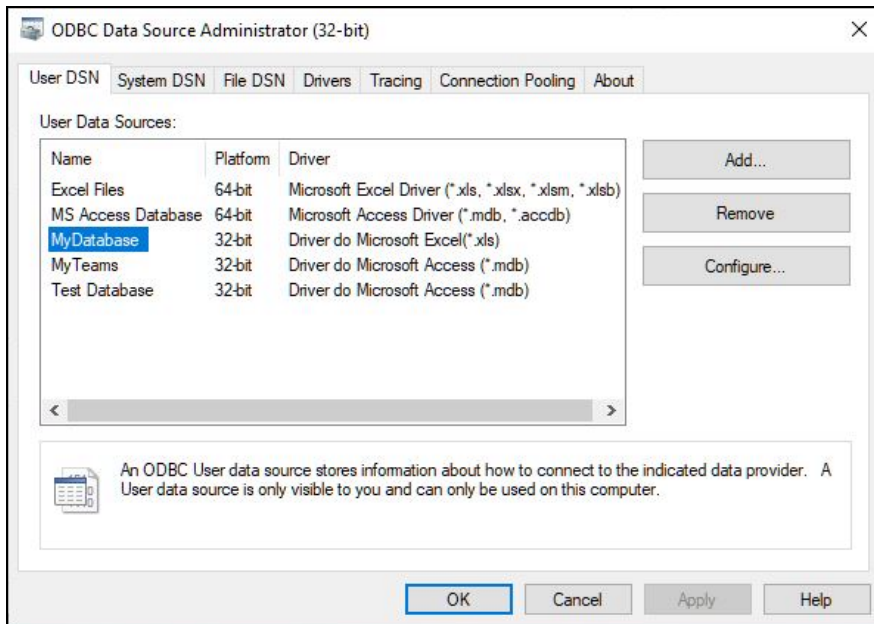
```
JsonGetArray(node1to2, MyKeys, ResultVar)
```


Database

ExcelRT can access a database with Open Database Connectivity (ODBC) script commands. ODBC is an international standard supported by most database vendors. With a few script commands, ExcelRT can read or write data in the database or present it in cells of a sheet.

ODBC gives your ExcelRT app access to MS Access, Visual FoxPro, FileMaker, Firebird, SQL Server, Oracle and others. You can even connect to a Microsoft Excel workbook although technically it is not a database. In Excel, each sheet is treated as a separate table where the column names in the first row are treated as fields of that table.

To access a database you will need an ODBC driver for that database that matches the 32-bit or 64-bit architecture of ExcelRT. The current build of ExcelRT on Windows is 32-bit and on macOS it is 64-bit. Most Windows computers include several preinstalled drivers. Some applications like Microsoft Office may add additional drivers. Free or low cost drivers can be downloaded and installed from the database vendor or third party software developers.



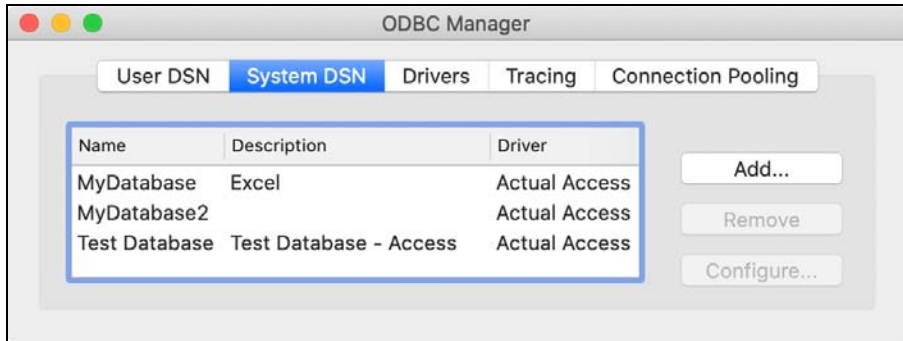
On Windows, there is a control panel that lists available drivers and allows a user to connect to a database by defining a DataSource name.

```
c:\Windows\SysWow64\odbcad32.exe
```

This command shows how to launch the control panel on Windows when the use clicks a button.

```
Button1=RunWinAppFromPath("c:\Windows\SysWow64\odbcad32.exe")
```

Apple no longer includes ODBC drivers with newer 64-bit builds of macOS. Third party developers like actualtech.com and openlinksw.com offer free or low cost solutions.



From either Mac or Windows, install an ODBC driver and add a DataSource string that allows the ExcelRT workbook to connect to that database.

This command connects to an ODBC database connection using DataSource and returns status in a variable named in ResultVar. The DataSource is a literal or variable containing the DataSource name already defined in the ODBC Admin app on Mac or Windows

```
DBconnect(DataSource,ResultVar,ExtendedSchema*)
```

ExtendedSchema is an optional parameter that can be TRUE or FALSE. Use FALSE for some databases and TRUE others. The results of the DBtables command may depend on the state of the ExtendedSchema parameter.

This command connects to an ODBC database connection using a set of credentials and returns connection status in a variable named in ResultVar. ExtendedSchema is an optional parameter that can be TRUE or FALSE.

```
DBconnectCredentials(DataSource,Host,Username,Database>Password,
ResultVar,ExtendedSchema*)
```

This command executes an SQL statement on the connected database. Use this command if the statement returns no output. The status is returned in a variable named in ResultVar.

One or more optional variable names can provide data that replaces a ? in the statement. The first ? is replaced with data from the first Var, the second ? with data from the send Var, etc.

```
DBexecuteSQL(Statement,ResultVar,Var1*,Var2*...)
```

Here is an example:

```
DBexecuteSQL(update customer set city=? where ZipCode=?,Result,City,Zip)
```

This command executes an SQL statement on the connected database. Use this command if the statement returns data into a RowSet from which specific data can be later extracted.

Some databases only allow the RowSet to be used with one extraction command. You must run the SQL command again to call another command using that RowSet.

The status is returned in a variable named in ResultVar. One or more optional variable names can provide data that replaces a ? in the statement. The first ? is replaced with data from the first Var, the second ? with data from the send Var, etc.

```
DBselectSQL(Statement,ResultVar,Var1*,Var2*...)
```

This command returns an Array of values from the specified column starting at index 1 in the active RowSet. The RowSet was previously created with a DBselectSQL command.

```
DBarrayRowSetColumn(Col,ArrayVar,ResultVar)
```

This command returns an Array of values from the specified row starting at index 1 in the active RowSet. The RowSet was previously created with a DBselectSQL command.

```
DBarrayRowSetRow(Row,ArrayVar,ResultVar)
```

This command returns the number of columns and rows in the active RowSet.

```
DBsizeRowSet(Columns,Rows,ResultVar)
```

This command stores the active RowSet into a JSON formatted string. A JSON formatted string can be stored in a JSON object. The JSON object can be accessed, modified or used by other commands. It returns SUCCESS or FAILED into a results variable.

```
DBjsonRowSet(StrVar,ResultVar)
```

This command returns an array of table names. The ExtendedSchema parameter in the DBconnect field may affect the results of the DBtables command.

```
DBtables(ArrayVar,ResultVar)
```

This command begins a transaction that can later be committed or rolled back. Some DB drivers do not support this command.

```
DBbeginTransaction(ResultVar)
```

This command commits a transaction. Some DB drivers do not support this command.

```
DBcommitTransaction(ResultVar)
```

This command rolls back a transaction. Some DB drivers do not support this command.

```
DBrollbackTransaction(ResultVar)
```

This command adds a row of data to the Table of a connected database. The ColumnList parameter provides a semicolon-separated list of column names. The status is returned in a variable named in ResultVar.

One or parameters provide the column values as a literal or variable name. Some ODBC database drivers are read only or may not support this command.

```
DBrowAdd(TableName,ColumnList,ResultVar,Col1*,Col2*...)
```

This command closes an open ODBC database connection and returns the status in ResultVar.

```
DBclose(ResultVar)
```

This command stores the active RowSet into a CSV structure. It returns SUCCESS or FAILED into a results variable.

```
DBcsvRowSet(CsvNumber,ResultVar)
```

This command stores the active RowSet into a sheet. It returns SUCCESS or FAILED into a results variable. Use TRUE in the optional Header parameter to include empty row at the top to later add column headers.

```
DBsheetRowSet(SheetID,ResultVar,Header*)
```

Deploy ExcelRT

Once your Excel workbook has been converted and refined to work well in ExcelRT, you have several options for distributing it to users as a free or commercial application.

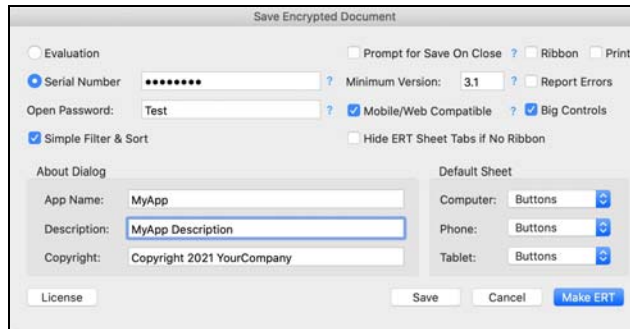
- **Unprotected XML File** – Distribute unprotected XML File. Instruct users to download ExcelRT from www.excelsoftware.com to use your file.
- **Protected ERT File** – Save a protected ERT file by purchasing a Serial Number for ExcelRT. Distribute ExcelRT with your file or instruct users to download ExcelRT from www.excelsoftware.com.
- **Standalone App** – Generate a protected Mac or Windows application with an optional splash screen, license agreement, custom user interface, custom icon and any type of Trial, Product, Try/Buy or Subscription license applied. ExcelRT can be distributed within your application so no separate install process is required.
- **No License** – Distribute an unlicensed, shared app on any platform without configuring Cloud License or requiring an activation server.

Generate ERT File

ExcelRT supports two file types, .xml and .ert. The .xml file is an XML formatted text file that can be opened and examined by a developer using either a text editor or with commands like Cell Edit on the ExcelRT **File** menu. The .ert file is an encrypted file that hides internal workbook data from a user.

ExcelRT is free for any user to download and use on any supported computer or device. Anyone can distribute XML files royalty-free.

If you want to protect distributed files, choose the **Save Encrypted** command from the **File** menu in ExcelRT. You will need to purchase a Serial Number for ExcelRT from Excel Software. That Serial Number grants to you unlimited distribution rights for any ERT files that you create.



Save Encrypted ERT File From XML File

To try your own .ert file on your own computer prior to purchasing a Serial Number, set the Evaluation radio button in the Save Encrypted Document dialog. Set the Mobile/Web Compatible checkbox to create an ERT file that can be used in a desktop or Cloud app.

Before distributing your workbook to customers, you may want to hide gridlines between cells or column and row headings or scrollbars. Use the Workbook Properties dialog in ExcelRT Builder to customize each sheet. Invisible sheets show up in ExcelRT as an unnamed tab that cannot be selected. Move invisible sheets in your Excel workbook to the right of visible sheets.

Use the Hide ERT Sheet Tabs if No Ribbon checkout to hide the sheet selector. This can be useful if you already provide navigation buttons on your sheets.

Standalone App

QuickLicense or AppProtect is used to deliver your workbook as a protected application that can be licensed to a specific computer.



QuickLicense supports many license types including time-limited trial, product, try/buy, subscription or floating licenses.

QuickLicense supports many activation processes including manual offline, online Serial Number activation and USB dongle activation when generating your own dongles with MakeDongle.

A screenshot of a software window titled 'Activation'. The window has a close button (X) in the top right corner. The text inside reads: 'This application requires an Activation Code.' followed by 'Request Number: 1661481472'. Below this, it says: 'When you receive your Activation Code, paste it in the field below and click the Activate Now button.' At the bottom, there is a text input field, a button labeled 'Activate Now', and a button labeled 'Activate Later'.

Manual Activation for Protected Application

QuickLicense can optionally be used with the Safe Activation service to automate online Serial Number activation. The vendor distributes a Serial Number to each customer during the purchase process that controls how many computers the application can be activated on.

A screenshot of a software window titled 'Activation'. The window contains a form with several input fields: 'Serial Number', 'First Name', 'Last Name', 'Company', 'Phone', and 'Email'. The 'Serial Number' field is currently selected with a blue border. At the bottom right of the window, there are two buttons: 'Activate Later' and 'Activate Now'.

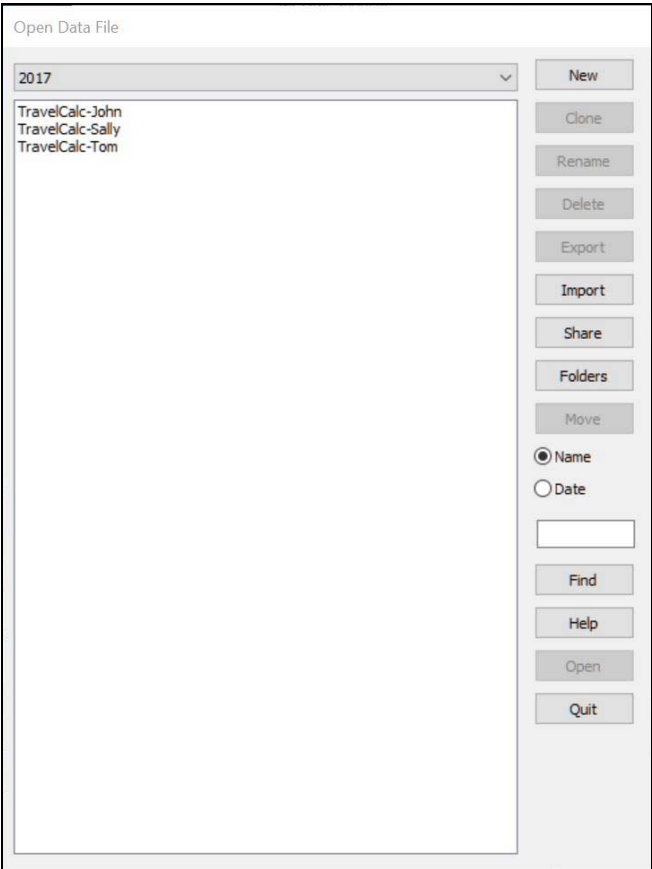
Online Serial Number Activation

The QuickLicense product includes the QuickLicense tool to define the license and generate a Ticket file. Use the AddLicense wrapping tool to generate a protected Mac or Windows application from an ERT and Ticket file.

The setup process gives a developer many optional features including a configurable Open Data File interface window, splash screen, pre-activation license agreement and a custom application icon.

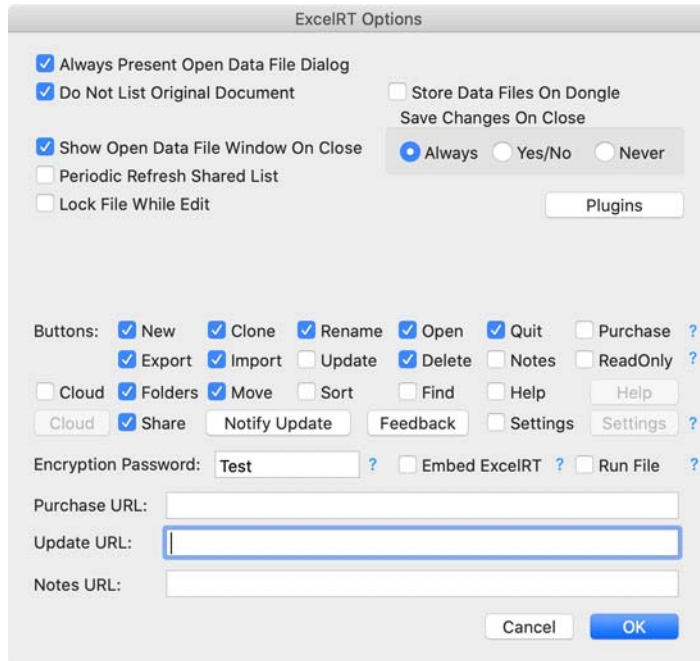


Excel workbooks are often client or project oriented. The user creates a different copy of the workbook to store unique data about one of their clients or projects. AddLicense implements an optional Open Data File window that gives your app a powerful user interface for this type of application.



Optional Open Data File Interface Window

The ExcelRT Options dialog controls the user interface presented by your protected application.

The image shows the 'ExcelRT Options' dialog box. It has a title bar 'ExcelRT Options'. Inside, there are several sections of settings. The first section has checkboxes for 'Always Present Open Data File Dialog' (checked), 'Do Not List Original Document' (checked), and 'Show Open Data File Window On Close' (checked). There are also checkboxes for 'Store Data Files On Doogle' (unchecked), 'Save Changes On Close' (unchecked), 'Periodic Refresh Shared List' (unchecked), and 'Lock File While Edit' (unchecked). A radio button group for 'Always', 'Yes/No', and 'Never' is present, with 'Always' selected. A 'Plugins' button is at the bottom right of this section. The second section is labeled 'Buttons:' and contains a grid of checkboxes for various actions: 'New' (checked), 'Clone' (checked), 'Rename' (checked), 'Open' (checked), 'Quit' (checked), 'Purchase' (unchecked), 'Export' (checked), 'Import' (checked), 'Update' (unchecked), 'Delete' (checked), 'Notes' (unchecked), 'ReadOnly' (unchecked), 'Cloud' (unchecked), 'Folders' (checked), 'Move' (checked), 'Sort' (unchecked), 'Find' (unchecked), 'Help' (unchecked), 'Share' (checked), 'Notify Update' (unchecked), 'Feedback' (unchecked), 'Settings' (unchecked). There are also buttons for 'Help', 'Feedback', and 'Settings'. The third section is 'Encryption Password:' with a text field containing 'Test' and checkboxes for 'Embed ExcelRT' (unchecked) and 'Run File' (unchecked). The fourth section has text fields for 'Purchase URL:', 'Update URL:', and 'Notes URL:'. At the bottom are 'Cancel' and 'OK' buttons.

Customize the User Interface of Protected Application

ExcelRT on Customer Computer

ExcelRT can either be installed first as a separate application on the Customer computer or together with your application. When installed separately, it must be installed to its default location.

- Mac - /Applications/ExcelRT
- Windows – c:\Program Files (x86)\Excel Software\ExcelRT

Excel Software recommends that you direct your customer to our ExcelRT installer or supply them with our installer file.

www.excelsoftware.com/excelrt-download

Mac Application Embed ExcelRT

On Mac, AddLicense can embed ExcelRT in the application bundle.

Mac or Windows Application

Alternatively, on Mac the ExcelRT folder that contains the ExcelRT.app can be included in the same folder that holds your application. On Windows you can also distribute the ExcelRT folder and its nested files in the same folder as your application.

For example, create a folder named Product (or give it your product name). Inside that folder put Product.exe (Windows) or Product.app (Mac) and the entire folder of ExcelRT files. Now zip the Product folder and put it on your website for download, on dropbox or send it to a customer as an email attachment. The ExcelRT folder must contain a Plugins folder with full read/write access for all user accounts.

```
Product
...Product.exe
...ExcelRT
```

Keep in mind that some email systems prevent you from sending or receiving some file types including .exe or .zip files. Giving a download URL to your customer is usually the best solution for distributing your finished product.

When your customer receives a zip file on Mac it is automatically unzipped into their **Downloads** folder. On Windows, they can **Right** click on the file and choose the **Extract All** command from the popup menu.

Your application can be installed on a customer computer with a Setup app created by a separate tool. On Mac, see ClickInstall from Excel Software to construct a professional Setup app.

ExcelRT Folder Location

When your Mac application is launched, it first looks for ExcelRT embedded in the application bundle, next it looks in the folder holding your application and finally it looks in the default location.

When your Windows application is launched, it first looks in the folder holding your application and if not found it then looks in the default location.

When distributing ExcelRT inside your Product folder, ExcelRT can generally be stored anywhere on the customer computer. The ExcelRT folder contains a Plugins folder that may need read/write access from the user account running your application. If your workbook uses scripting commands that write data to the Plugins folder, then the user account must have read/write access to files in the Plugins folder. A safe place to store your Products folder is inside the Documents folder on the Mac or Windows computer.

Activate Standalone App

When a user launches your protected application for the first time, an activation process occurs on the user computer.

For a manual activation process, an Activation dialog is presented that collects an Activation Code. The user gives you a Request Number shown in the dialog, you enter it into QuickLicense and give them the computer unique Activation Code needed to activate the app on their computer.

For an online activation process, you give the user a Serial Number at the time of purchase. The user enters the Serial Number into the presented Activation dialog and clicks the **Activate Now** button to complete the activation process.

Refer to Tutorial 1 or Tutorial 2 included with QuickLicense for step-by-step instructions on how to configure a license with manual or online activation.

As a developer, it is important to understand what happens during the activation process on the user computer. Regardless of the QuickLicense configured license type and activation process, a shared Ticket folder is created if it does not exist.

- On Windows: `c:\users\public\ticket`
- On MacOS: `/users/shared/ticket`

Two files are generated in the Ticket folder, the active Ticket file and a Build file that you'll recognize because it contains your application name and a long number.

Once activated, the user is not prompted with the Activation dialog on future application launches. During the development process you may want to present the Activation dialog again for testing or to create screen shots for user documentation.

For a simple product license that allows reactivation, simply delete the Ticket and Build file from the shared Ticket folder. Now launch the application to see the Activation dialog again. For a time-limited dialog that does not allow reactivation, refer to the manual reset process in the QuickLicense User Guide.

The activation process also creates a folder on the user computer that holds data files. If the Open Data File window is used, a new file is created in this folder each time the user clicks the **New** or **Clone** button.

Assume the app is named `TravelCalc.exe` on Windows or `TravelCalc.app` on Mac. These files contain the user-entered data, so they may want to keep backups.

- On Windows: `c:\users\public\TravelCalc`
- On MacOS: `/users/shared/TravelCalc`

The master copy of your ERT file is also stored in this folder, but it may not be visible and it isn't shown in the Open Data File window. The **New** button in that window simply makes a named copy of the master file.

Update Standalone App

Assume you have been selling your protected ExcelRT product to customers. You have an installed base of users that have created many files of user-entered data.

Assume you have created a new ExcelRT file with major improvements that you want to make available to your existing customers.

Within AddLicense, set the Retain Activation dialog. This allows an existing customer to switch to your new application without requiring them to activate again.

Present the ExcelRT Options dialog. Learn about and enable the **Update** button. Upload your new ERT file to your website and enter its URL into the Update URL field of the ExcelRT Options dialog. Now build the new EXE or APP.

New users can download the new EXE or APP file like before. Existing user can optionally download the new EXE or APP file, but they will also need to click the **Update** button. This replaces the master ERT file on their computer with the file stored on your website.

If you already had the Update process setup in your original application, then existing user will simply click the **Update** button. There is no need to download the application again.

If you don't have a website, instruct the user to drag your newly supplied ERT file to the appropriate folder on their computer.

Deploy Plugin Files

On Mac or Windows, a human or installer program can add files to the ExcelRT Plugins folder. To simplify the user experience, this section describes how to deploy Plugin files with a Standalone application.

Standalone ExcelRT App

Set the Embed Static Files checkbox in AddLicense. Add a nested folder within the Source folder that holds the ERT file and includes all required Plugin files. Choose the ERT file with the **Select** button to the right of the Application field.

On first launch of a standalone Mac or Windows application, the Plugin files will be extracted from the application and added to the appropriate Plugins folder. If ExcelRT is located in the same folder as the application itself, files are added to the Plugins folder also in the same folder as the application itself.

If your standalone application does not include its own copy of ExcelRT, then ensure the user installs ExcelRT to the default location before they install your application. Here is the default location of ExcelRT.

- Windows – c:\Program Files (x86)\Excel Software\ExcelRT
- Mac - /applications/ExcelRT

If your application has already been installed and you later build a new application that includes additional Plugin files, you may need to force a new activation on the User's computer.

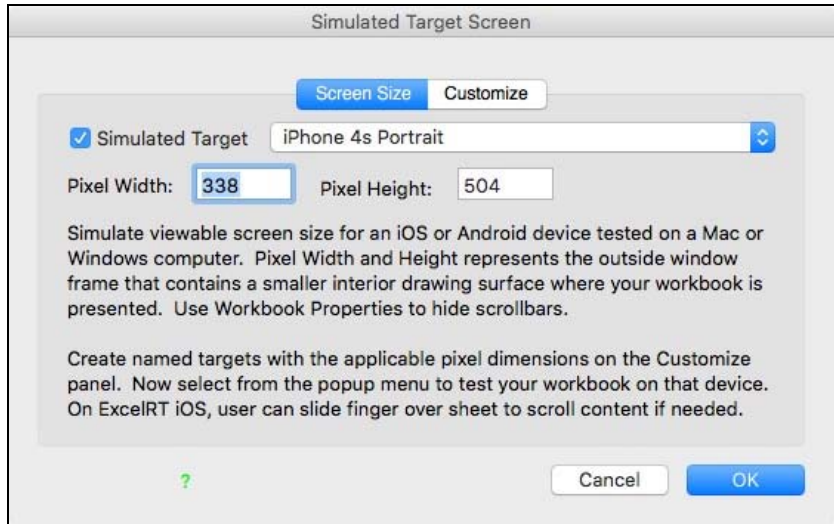
Ask the user to launch the application while the **Shift**, **Control** and **Command** keys are pressed. The Apply New License dialog is presented. Click **Yes** to dismiss the dialog. Now launch again without the keys pressed and perform a new activation.

Alternatively, instruct the user to delete the Ticket and all Build files for your application from the shared ticket folder location before reinstalling.

- Windows – c:\users\public\ticket
- Mac - /users/shared/ticket

Screen Size and Orientation

In Design mode, ExcelRT has a Simulated Target Screen dialog that allows your workbook to be viewed on different screen sizes and orientations. This feature is intended for future use when designing or testing mobile Apps.



View Workbook Sheets on Different Screen Sizes

The Pixel Width and Height field value will determine the Window size that controls the viewable drawing surface within the window.

Use the Customize panel to add your own device names and calibrated pixel values. This dialog allows you to use any Mac or Windows computer for a screen simulation of any current or future target hardware.

If your sheets are designed for a specific window size even when running on desktop computers, then leave the Simulated Target checkbox set when creating and ERT file. This presents your workbook within a non-resizable window on desktop computers.

Purchase Button

A **Purchase** button can be added to the Open Data File interface window. Set a checkbox in the ExcelRT Options dialog in AddLicense when building a standalone or shared product. The **Purchase** button presents a dialog that allows a user to make InApp purchases with a Credit Card or Paypal account.

All information presented by the Purchase dialog is configured from the Purchase Button Edit page accessed from the Vendor Info page on a Safe Activation Service 3 account.

The screenshot shows two overlapping windows. The background window is titled 'Open Data File' and has a sidebar with buttons: New, Clone, Rename, Delete, Export, Import, Open, and Purchase (highlighted in blue). The foreground window is titled 'Purchase' and contains the following elements:

- Instruction for the user
- Form fields for 'First Name' and 'Last Name'.
- A table with the following data:

Name	Quantity	Price	Shipping	Tax	Subtotal
Purchase Order Report	0	100.00	0.00	0.00	0.00
Materials Report	0	50.00	0.00	0.00	0.00
User Guide	0	19.95	0.00	0.00	0.00

Below the table, it shows 'Stored Card: 1125' and 'Saved Until: 2018-04-30'. On the right, it says 'Total: 0.00 USD'. At the bottom are buttons for 'Credit Card', 'Pay with Card', 'Click for Details', and 'Done'.

Purchase Dialog Presented from Open Data File Dialog

A **Purchase** button is only available for a license created with QuickLicense. When configuring that license, you will identify the Product ID in Safe Activation associated with that license. You'll need that Product ID to configure the Purchase process within Safe Activation.

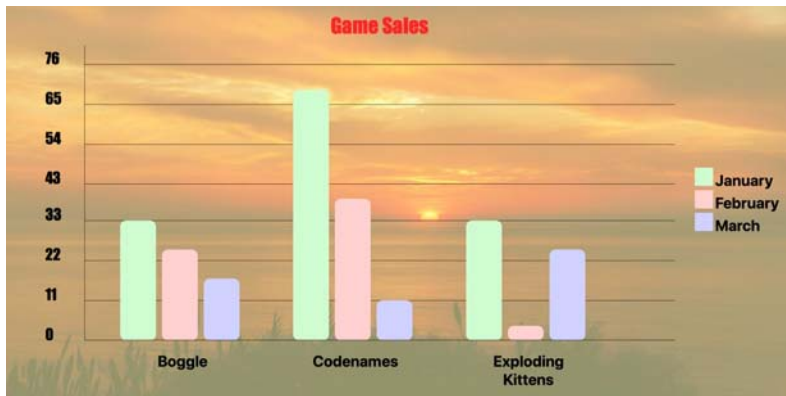
The process used to configure a Purchase dialog is demonstrated in several videos. Alternatively, a **Purchase** button can be linked to a page on your own website that is presented to the user in their default web browser.

Chapter

6

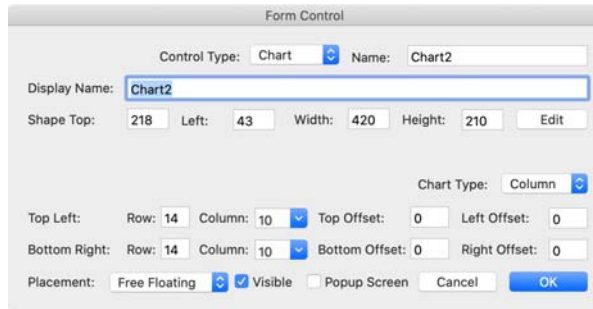
Charts

ExcelRT supports a suite of dynamic charts to present a table of data in a graphically pleasing format. Most of the familiar chart types from Microsoft Excel are supported with flexible style and customization options.



The data to build a chart comes from a range of cells. A chart is configured as a form control. A chart can be presented on any sheet with the underlying data coming from the same or a different sheet. When the underlying data changes, the chart is updated.

Within ExcelRT Builder, click the Form Control tool on a sheet, choose Chart as the Control Type and select the Chart Type from a popup menu.



Form Control

Control Type: Chart Name: Chart2

Display Name: Chart2

Shape Top: 218 Left: 43 Width: 420 Height: 210 Edit

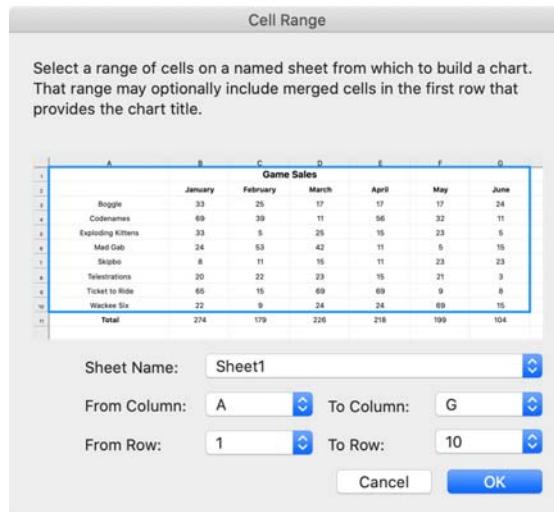
Chart Type: Column

Top Left: Row: 14 Column: 10 Top Offset: 0 Left Offset: 0

Bottom Right: Row: 14 Column: 10 Bottom Offset: 0 Right Offset: 0

Placement: Free Floating Visible ☐ Popup Screen Cancel OK

Click the **Edit** button to select the range of cells and select the chart style. The Cell Range dialog is used to choose the original range of cells displayed in the chart. This dialog is only presented if a cell range has not already been defined.



Cell Range

Select a range of cells on a named sheet from which to build a chart. That range may optionally include merged cells in the first row that provides the chart title.

	A	B	C	D	E	F	G
1		Game Sales					
2		January	February	March	April	May	June
3		Boggle	33	25	17	17	24
4		Codenames	69	39	11	56	32
5		Exploding Kittens	33	5	24	19	23
6		Mad Ops	24	53	42	11	5
7		Scopio	8	11	15	11	23
8		Teletations	20	22	23	15	21
9		Ticket to Ride	65	15	69	69	9
10		Wreckin' Six	22	9	24	69	15
11		Total	274	179	226	218	199

Sheet Name: Sheet1

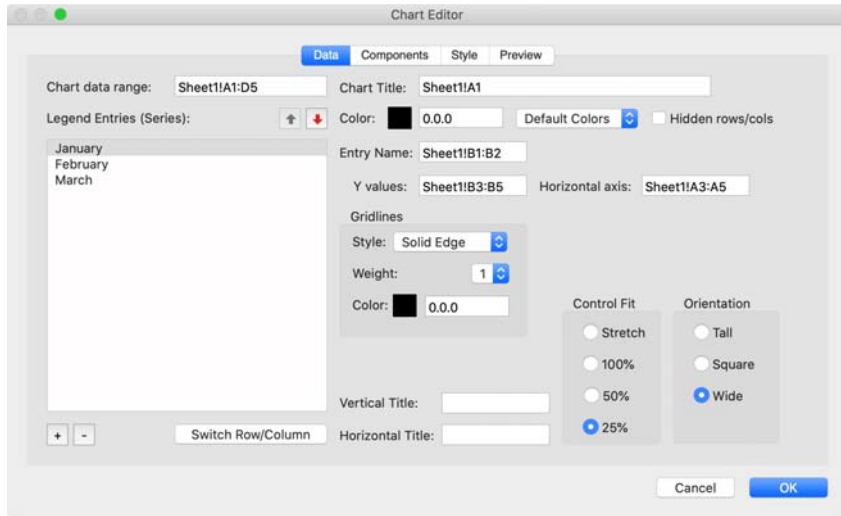
From Column: A To Column: G

From Row: 1 To Row: 10

Cancel OK

For most chart types, you will choose a range that includes the Column and Row headers as illustrated above with range A2:G10. If the sheet includes a table name, you can include that row in the range. Typically the table name will appear centered in a merged row of cells.

After setting a cell range, the Chart Editor dialog is presented. The Data panel of this dialog determines what cells of data are used to build the chart. One or more series of data values are derived from the data range. The Style panel offers a quick way to choose a predefined chart appearance, while the Components panel allows individual parts of the chart to be customized. Use the Preview panel to see the affects of changes in the Data, Components and Style panels.



The style of a chart is constructed from several parts. The chart type (Column, Bar, Area, Pie, Stacked Area, Stacked Column and Scatter) determines the content of the Plot area while the Style for a specific chart type determines the size, orientation or visibility of other parts.

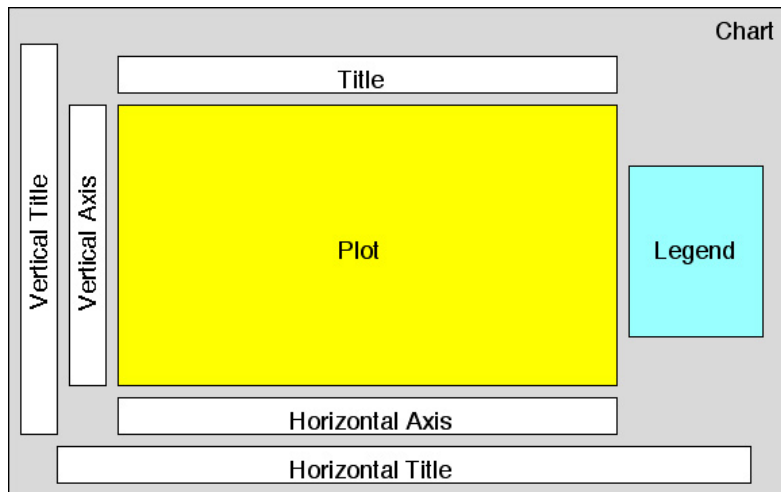


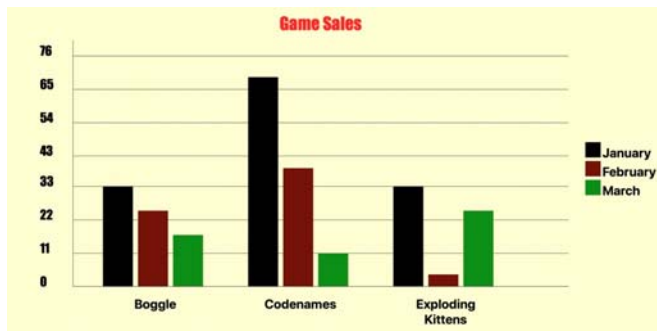
Chart Types

Specify a range of data values in a sheet and select an appropriate chart type. The chart type is the first and most important decision when constructing a chart.

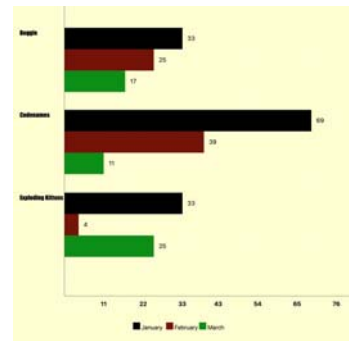
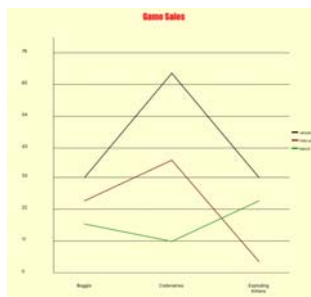
Chart type determines how data is presented in the Plot area of a chart. This range of data will be presented in various chart types below.

	Game Sales		
	January	February	March
Boggle	33	25	17
Codenames	69	39	11
Exploding Kittens	33	4	25

This Column chart type shows three series of data values (January, February and March) for each run (Boggle, Codenames and Exploding Kittens). With the Chart Editor dialog, a designer can change the horizontal and vertical axis with a button click.



The Bar chart type is similar, except the bars are horizontal. A Line chart uses lines instead of columns to show values.

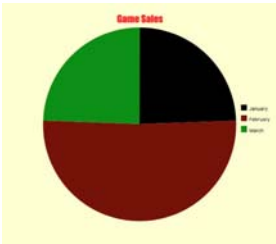


An Area chart uses a different color for each Series of data drawn with some transparency so the user can see each overlapping area.



This type of chart works best with a small amount of data.

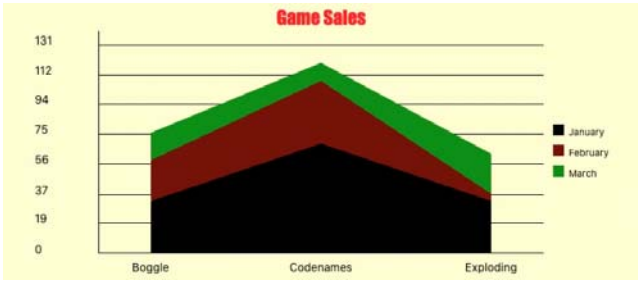
A Pie chart displays a single series of data points. It also works best with a small amount of data that highlights the proportion each part contributes to the whole.



A Stacked Column chart adjusts the vertical axis to the sum of each run in the series of data points.



A Stacked Area chart shows the sum of each run on the vertical axis and a stacked set of areas.



A Scatter chart is different than most chart types. It starts with one or more X and Y pairs in the range of data. In this example, the first two columns presents X and Y values for the first series of points, while the next two columns create the second series of points and so on.

S1 X	S1 Y	S2 X	S2 Y	S3 X	S3 Y
5	25	1	2	1	3
6	36	2	4	2	6
7	49	3	6	3	9
8	64	4	8	4	12
9	81	5	10	5	15
10	100	6	12	6	18

The Scatter chart determines the appropriate Vertical and Horizontal axis values based on the plotted data. It shows each point as a round or rectangular dot.

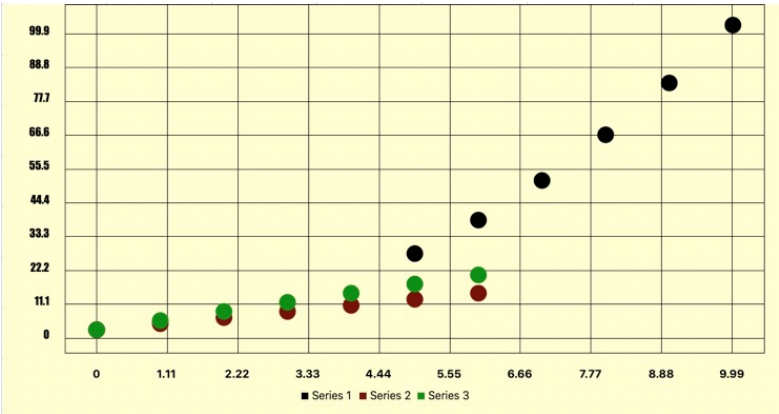
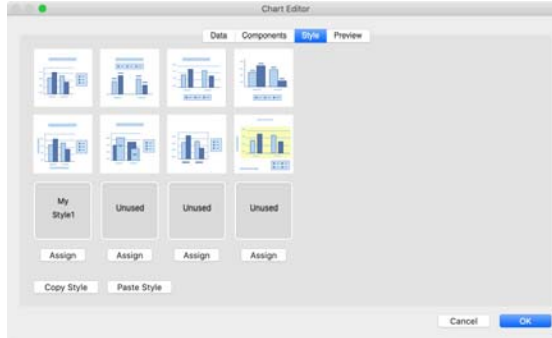


Chart Styles

Based on the selected chart type, the Styles panel of the Chart Editor shows predefined styles.

Click each icon on the Style panel and select the Preview panel to see your live chart. Once you have a good starting point, you can customize the chart with options on the Data and Components panel.



After customizing a chart style, you might want to name and save it so that custom style can later be applied to a different range of data to construct a new chart. Click the **Assign** button and the Custom Chart Style dialog is presented to name your custom style.



Custom chart styles are saved in a preference file (named ExcelRT.ini) associated with ExcelRT Builder. Most ExcelRT developers use AppProtect or QuickLicense to wrap the finished ExcelRT workbook into an App. A preference file for each tool is stored in the same folder location.

Use custom styles on any project by simply clicking on a named Custom Style icon in the Styles panel of the Chart Editor. You can also **Copy** and **Paste** style data as clipboard text between charts within the same or in different ExcelRT files.

Chart Components

Dozens of options are available to configure each part of a chart. A predefined style sets all the options with one click. The chart type determines the content of the Plot area while the style determines the size, orientation and visibility of other parts like the Title, Vertical and Horizontal Axis, etc.



Each part of a chart is positioned based on the X and Y percentage on the Components panel with the Width and Height also defined as a percentage. These parameters are integer numbers from 0 to 100. Some parts are unchecked if the style doesn't display that part.

Text for the Vertical and Horizontal Title fields come from the Vertical Title and Horizontal Title fields on the Data panel. That field can include the actual text or a cell reference of the form Sheet2!H8 that contains the actual text.

Text drawn in each part of a chart will use a specified Font, Size, Color, Alignment and bold or plain style. If the local computer does not have the specified font installed, it will use the system font.

To set the color of a property, click on the color box and choose a color from the present color wheel. The RGB representation of that color is shown as text next to the displayed color box. It is often easier to copy and paste the RGB text representation to ensure exact colors for different parts.

The color field for Chart Background, Plot Background and Legend Background can be empty for a transparent affect where the underlying sheet image is visible. Specify a color in the Chart Background field for an opaque presentation.

A chart may include a picture drawn on top of the other parts that is positioned and sized with the X, Y, Width and Height fields. Set the Background checkbox to use the picture as a background image. The Transparency field determines if the picture is opaque or somewhat transparent. Together with a Chart Background color, a variety of affects can be achieved.

Some chart types like column and bar charts use bars to represent data values. Bar properties affect the shape, color and width of bars and space between each bar. The meaning of these parameters can vary based on chart type.

Some chart types allow data labels to be placed within the Plot area of the chart.

When drawing columns, bars or areas in the Plot portion of a chart, the color of each series is defined on the Data panel. The designer can select each Series in the dialog and individually assign a color. Alternatively, choose a command on the Default Colors popup menu to assign a color theme to all series items.

Many chart types allow a grid within the Plot area. Use the Gridline properties on the Data panel to affect the color, weight and line styles of the grid.

Chart Dimensions

The data values presented in a chart will dynamically affect the chart dimensions. When constructing a chart, the Plot area is constructed first based on the number of rows and columns of data and the assigned bar properties. The other parts of the chart are then proportionately sized based on the percentage indicated in the Width and Height properties of that part.

The Orientation field on the Data panel allows a designer to provide guidance during the chart building process to minimize the affects of stretching or compressing text. Once the chart is constructed into a picture, that image is displayed on the sheet based on the Control Fit parameter.

There are many variables that affect the presentation of a chart. If names in a legend are too long or the specified text size is too large, the text may get clipped off or not be displayed at all. Some adjustments may be required to the position, width and height of the Legend part for a nice presentation.

Chart Data

The data for a chart comes from cells in a sheet. By default, the initial cell range specified by the designer determines how many series and runs of data are displayed. Invisible rows or columns in the chart range can be displayed or hidden based on the Hidden rows/cols checkbox on the Data panel of the Chart Editor.

The **Switch Row/Column** button switches the cell values used to create a series and used to create a run within that series. That concept is easier to see then explain by simply clicking the button and looking at the Preview panel.

Use the **Up** and **Down** buttons to change the order of each series of data in the chart. Add or remove a series of data with the + and – buttons.

Data for a chart typically comes from a contiguous range of cells on one sheet, but that is not a requirement. In an unusual situation, a designer might add a series of data using a range of cells from a different sheet.

Given all of the configurable options for where data comes from and how it is presented, it is possible to create an invalid chart. If that happens, the chart image displays Bad Chart Setup in a rectangular box. During design, an invalid cell reference could cause an exception in ExcelRT when it attempts to draw the chart.

Build and Use Charts

A chart is a type of form control that is added, deleted, moved or resize with the form control tools in the ExcelRT Builder ribbon.

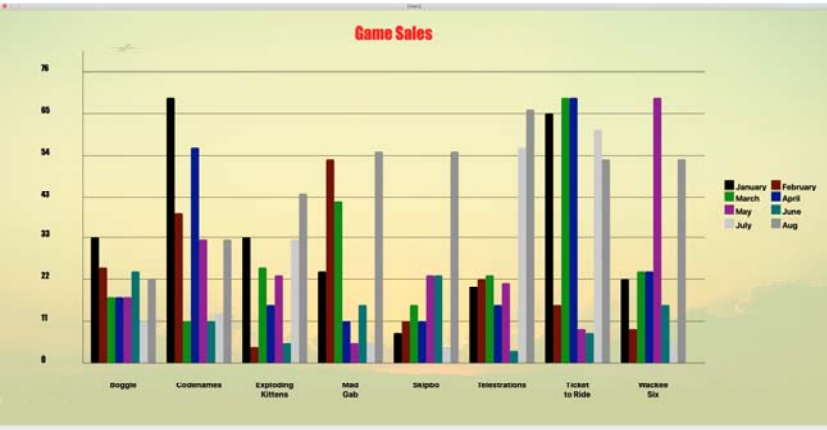


Select the **Control Add Delete** tool and click on a sheet to add a control, then select type Chart. To delete an existing chart, hold down the **Shift** key and click on the chart with that tool.

To edit an existing chart, click on it with the **Control Edit** tool. Alternatively, while is ExcelRT Builder mode the designer can simply click on the chart with the arrow cursor.

The **Control Move Size** tool is used to move the top left corner of a chart to a different location on the sheet. The chart size can be set with the Width and Height properties in the Form Control dialog. Alternatively, the displayed chart size can be determined dynamically as a fixed percentage of the actual chart size that depends on the amount of data and properties of the chart. See the Control Fit parameter on the Data panel of the Chart Editor dialog.

In the finished application, the designer may want to show a compressed image of the full chart. The user of the finished application can click that smaller image to present a full sized chart in a popup window. To enable this feature, set the Popup Screen checkbox in the Form Control dialog.

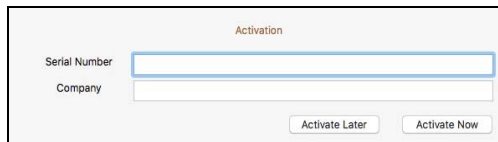


Chapter 7

ExcelRT Builder

ExcelRT Builder can be enabled on Mac or Windows by entering a Serial Number for the ExcelRT Annual Subscription. The same ExcelRT installation used by a customer can also serve as the Builder for a developer. A Serial Number allows the Builder to be active on one computer at a time.

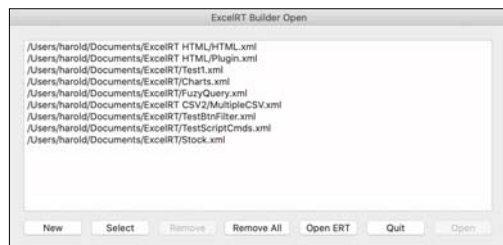
Launch ExcelRT and choose the **License** command from the **File** menu.

The image shows a dialog box titled "Activation". It contains two text input fields: "Serial Number" and "Company". Below these fields are two buttons: "Activate Later" and "Activate Now".

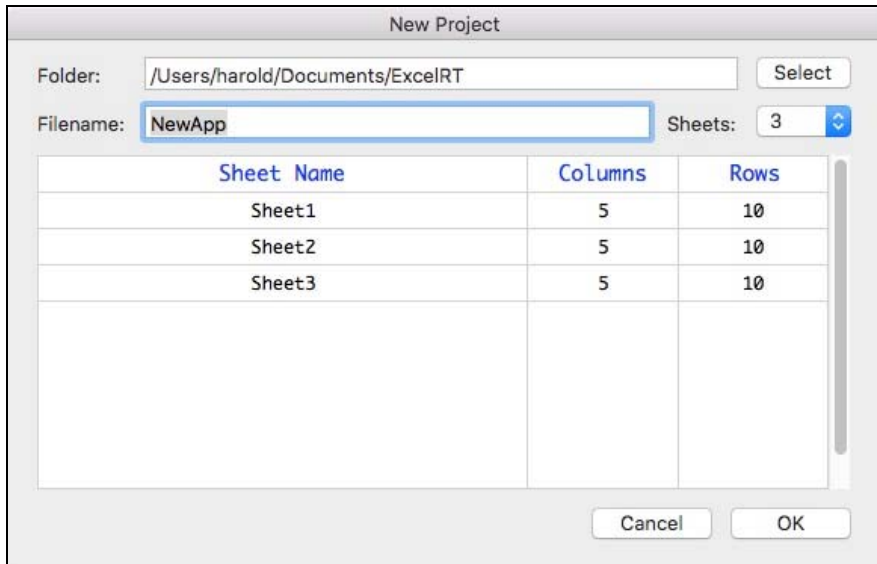
Activate ExcelRT Builder

After activation, quit and restart ExcelRT. Notice the ExcelRT Builder Open dialog is presented.

Click **New** to create a project, click **Select** to choose an existing XML file to add to your project list or select a project from the list and click **Open**.



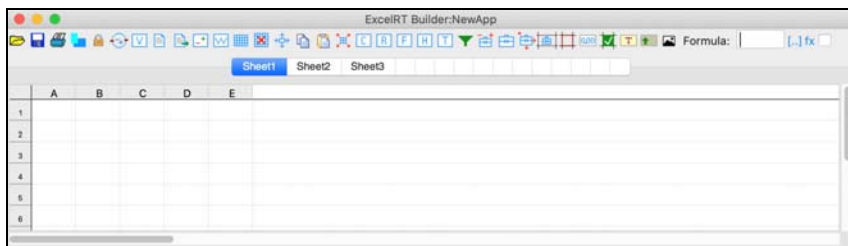
The **New** button in the ExcelRT Builder Open dialog presents the New Project dialog. Use the **Select** button to locate the folder path where the project is stored. Enter the project Filename that will automatically be saved with file extension XML.



Create a New Project

Select the number of sheets in the project, then select and rename each sheet as desired. Select and change the number of Columns or Rows in each sheet, then press **Enter**. Click **OK** to create and open the project.

ExcelRT Builder adds a toolbar at the top of the main window.



ExcelRT Builder Window

Builder Tools

The Builder Toolbar contains all the tools needed to author an ExcelRT workbook from which an application is generated.



Open – Click to open a new project.



Save – Click to save project changes.



Print – Click to print.



Simulated Screen - Click to view workbook on different screen sizes and orientations when designing for mobile phones and tablets.



Save Encrypted - Click to save the current XML project, then export to a new ERT file. The ERT file is then opened with ExcelRT Builder features disabled.



Recalculate Cells – Click to recalculate all cells on all sheets.



Variables – Click to add, edit, view or delete Script variables.



Script Edit – Click to present the Script Editor. Hold down the **Shift** key while clicking to present a plain text editor.



Script Run – Click to run script for a simulated button click.



Sheet Add Delete – Click to add, delete or reorder sheets in the workbook. Be careful when reordering sheets since Script commands that refer to a specific sheet ID will no longer work and may crash ExcelRT.



Workbook Properties – Show or hide sheet column and row headers, scrollbars and gridlines. Rename sheets or make them Visible or Invisible. The default window size of the workbook when initially opened can be set here.



Sheet Size – Change the number of Columns and Rows on the current sheet.



Erase Cell Data – Select a Column and Row range for which to delete cell properties. By default, all cell properties are selected for deletion. Selectively delete specific cell properties by clearing checkboxes.



Cell Size & Visibility – Change the Column Width, Row Height or visibility of columns and rows.



Cell Range Copy – Copy selected properties for a range of cells into the clipboard.



Cell Range Paste – Paste selected cell properties copied into clipboard to a range of cells in the current sheet. Alternatively, generate sample text, integer or number data into a range of cells.



Cell Range Move – Click to move a cell range with related properties on the current sheet.



Cell Properties – Click to edit cell properties for a selected cell. If no cell is selected, the Select Cell Properties dialog is presented to choose a range of cells for which properties can be modified.

To unselect a selected cell, first click to the top left of the Row and Column header, then click Cell Properties to present the Select Cell Properties dialog. Hold down the **Shift** key while selecting the tool to suppress the Cell Properties dialog.



Range – Click to add, edit, delete or view named ranges.



Format Rules – Click to add, edit, delete or view formatting rules.



Hyperlinks – Click to add, edit, delete or view Hyperlinks assigned to a range of cells.



Table Add Delete – Click to add, edit or delete tables within the project. Tables overlay extra properties on a range of cells.



Sheet Filter – Enable or disable a sheet filter for a range of cells on a sheet.



Control Add Delete – Click to select tool, then click on screen to add a Form Control. The Add Form Control dialog is presented to define the control name, type, width and height. To delete an existing control, hold down the **Shift** key while clicking the tool on the existing control.



Control Edit – Click to select tool, then click on an existing Form Control on the screen to edit it. Double-click on the Control Edit tool to present the Form Controls dialog to Add, Edit, Delete or view any form controls in the project.



Control Move Resize – Click to select tool, then click in the top left quadrant of an existing form control to get the Green Resize cursor. Click on the screen position where you want the control to be top-left aligned. Click the tool in the bottom right quadrant of a form control to resize it using a Red Resize cursor to choose the bottom right position. Pictures cannot be resized using this process.



Cell Control – Click to select tool, then click on any cell to create an inline cell control. A cell control is drawn within a specific cell and provides an alternative to form controls.



Cell Borders – Click to select tool, then click on any cell to assign borders.



Cell Validation – Click to select tool, then click on any cell to set entry validation rules, input messages and alert messages.



Cell Text Color – Click to change font, size, color, style and other text properties of a cell.



Cell Background – Click to change background color and pattern for a cell or add an icon.



Sheet Background Image – Click to add or remove an image used as a background for the current sheet. Each sheet can have a different background image or typically, no image at all.

Formula – Edit the formula for the selected cell and press **Enter** to store that formula in the cell. A formula begins with the = character.

[...] – Click to present a dialog used to edit a long formula.


Fx – Click to present a library of support Excel functions. When a function is alphabetically located and selected, it gets inserted into the formula edit field at the cursor location.

Rebuild Dependency – The Rebuild Dependency checkbox is an unnamed checkbox to the right of the Formula selector.



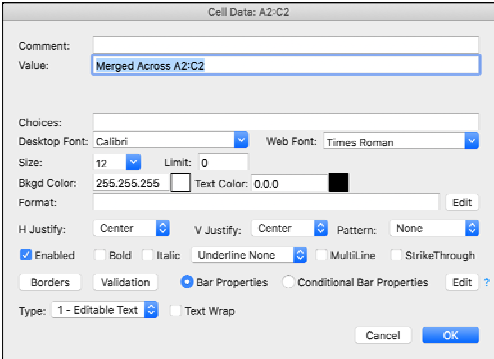
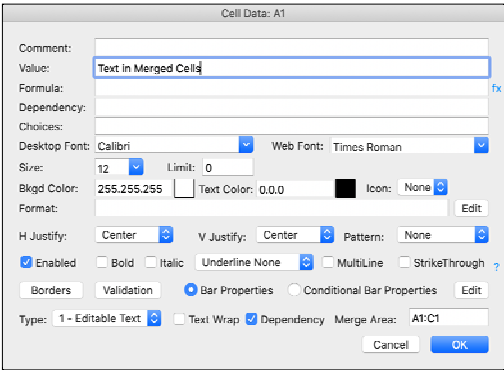
When checked, all cell dependencies are recalculated whenever any formula is changed. To improve editing performance in a large workbook, clear this checkbox and choose the **Build Dependency** command from the **File** menu when all editing is completed.

Cell Properties

 Click on a cell with the Cell Properties tool to present the Cell Properties dialog. This dialog shows all the properties related to a specific cell. The cell being edited is displayed in the title of the dialog.

It is often more convenient to edit some of this data in other ways. For example, text can be typed directly into a cell on the screen. The formula can be edited in the Formula field at the top right of the window. Cell color and formatting is often assigned with other tools.

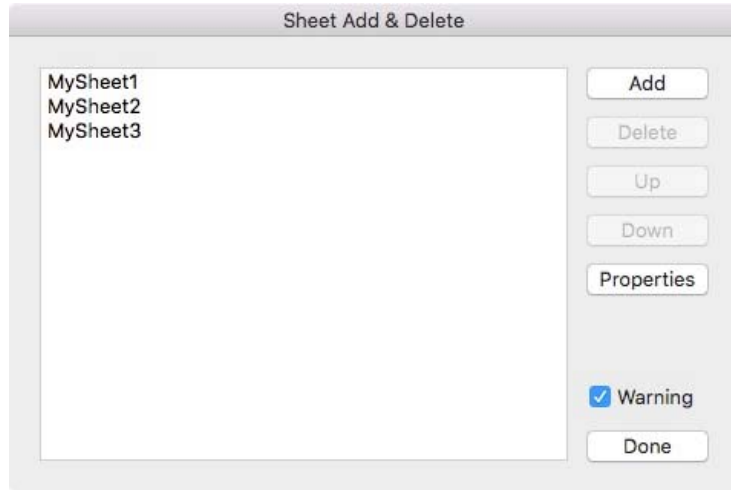
When clicking the Cell Properties tool, the Select Cell Properties dialog is presented. This dialog allows the developer to select a range of cells and then edit most of the cell properties within all cells in that range.



Sheet Add Delete



Present the Sheet Add & Delete dialog to add, delete or reorder sheets in the workbook. A workbook may contain up to 15 sheets and each can be visible or not.



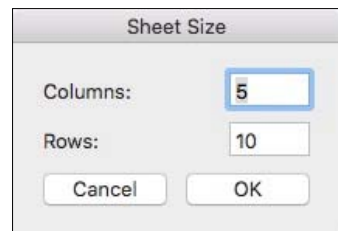
Click the **Add** button to add a new sheet. The sheet is given a default name. Click the **Properties** button, then select and change the Sheet title.

If the Warning checkbox is set, you will be warned whenever reordering or deleting sheets since that changes the index of a sheet. The sheet index might be referenced from a Script command.

Sheet Size



Present the Sheet Size dialog to change the number of Columns and Rows in the current sheet. When adding new columns or rows to a sheet, all the new cells have no data or assigned properties.



Erase Cell Data



Present the Erase Cell Data dialog to remove all or selected cell data from a range of cells.

When you erase Text Color you are assigning it to default black and erasing Bkgd Color sets it to white.

If Font and Size is checked, the default font and size is applied.

Use the Cell Properties and Erase Cell Data dialogs over a range of cells to assign the appropriate data to each cell.

Cell Size & Visibility



Change the Column Width, Row Height or visibility of columns and rows.

To assign a specific Cell Type to a range of cells, select the starting and ending column and row. Set the Cell Type checkbox and choose a Cell Type from the popup menu.

Format Rules

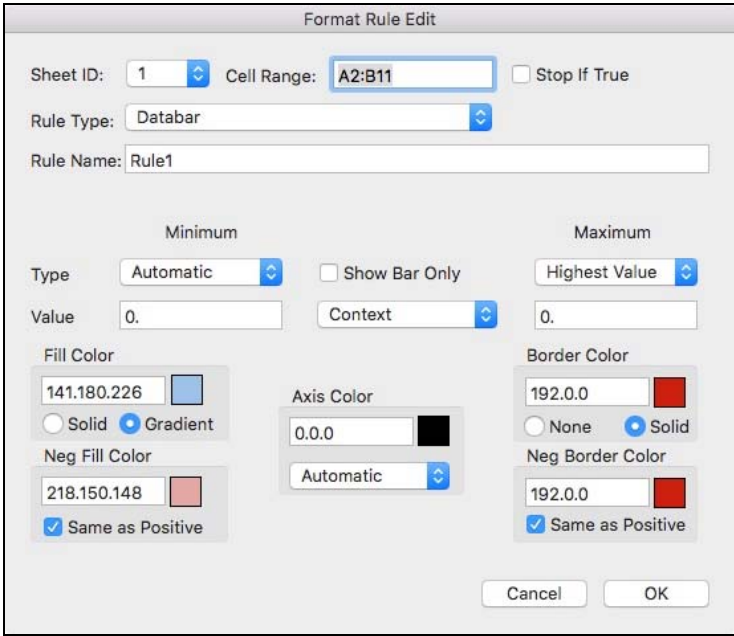


Click to add, edit, delete or view formatting rules in the Format Rules dialog.

A formatting rule consists of a complex collection or raw data that is displayed if the Rule Name is unchecked or if you click the **Info** button for a selected rule.


Use the Format Rule Edit dialog to add or edit a rule. Start by selecting the Rule Type to customize the dialog to the fields needed to define that specific type of rule.

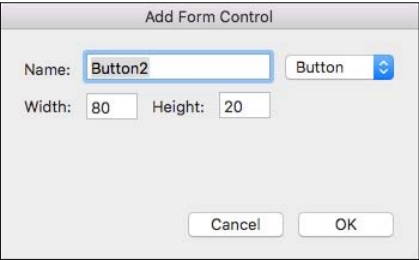
ExcelRT implements approximately the same set of formatting rules and options as those implemented by Microsoft Excel.

The 'Format Rule Edit' dialog box is shown. It has a title bar 'Format Rule Edit'. Inside, there are fields for 'Sheet ID' (set to 1), 'Cell Range' (set to A2:B11), and a checkbox 'Stop If True'. Below these is a 'Rule Type' dropdown set to 'Databar' and a 'Rule Name' text field containing 'Rule1'. The main area is divided into 'Minimum' and 'Maximum' sections. Under 'Minimum', there's a 'Type' dropdown set to 'Automatic', a 'Value' field set to '0.', and a 'Context' dropdown. Under 'Maximum', there's a 'Type' dropdown set to 'Highest Value' and a 'Value' field set to '0.'. There are also checkboxes for 'Show Bar Only' and 'Neg Fill Color'. Below these are color selection areas for 'Fill Color' (with a color picker showing 141.180.226 and a 'Gradient' radio button selected), 'Axis Color' (set to 0.0.0), and 'Border Color' (set to 192.0.0 with a 'Solid' radio button selected). At the bottom right are 'Cancel' and 'OK' buttons.

Databar Formatting Rule Setup

Control Add Delete

 Click to select tool, then click on the screen to add a Form Control. Form controls are used in Microsoft Excel to add push buttons, checkboxes and radio buttons to a sheet. The Add Form Control dialog is presented to define the control name, type, width and height.

The 'Add Form Control' dialog box is shown. It has a title bar 'Add Form Control'. Inside, there's a 'Name' field set to 'Button2' and a dropdown set to 'Button'. Below these are 'Width' and 'Height' fields set to '80' and '20' respectively. At the bottom are 'Cancel' and 'OK' buttons.

Based on the selected control type, additional fields will be presented in the dialog. For some control types, you may need to edit additional parameters by presenting the Control Edit dialog for the newly created control.

Control Edit



Click on a form control with the Control Edit tool to present the Form Control dialog. The fields in the dialog will depend on the selected control type. The next most important selection is the Placement field.

Form Control

Sheet ID: 4 Control Type: Button Name: Button 1

Display Name: Export Data

Shape Top: 95 Left: 25 Width: 87 Height: 19

Top Left: Row: 7 Column: 1 Top Offset: 91 Left Offset: 0

Bottom Right: Row: 8 Column: 2 Bottom Offset: 105 Right Offset: 70

Placement: Move And Size ☒ Visible Cancel OK

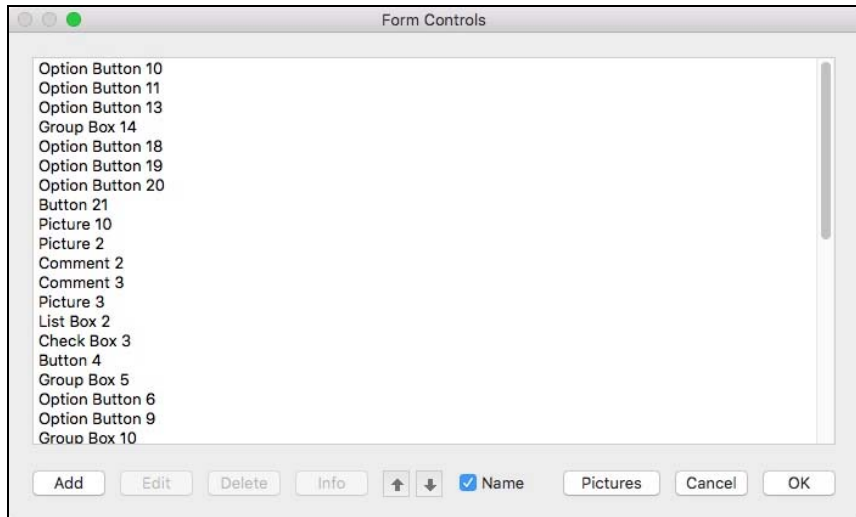
If Placement is Free Floating, the Shape Top and Left fields determine the location of the top left corner of the control on the selected sheet. The Width and Height fields determine the control size. A developer will typically use the **Control Move Resize** tool to indirectly affect these fields.

If Placement is Move or Move And Size, the control is aligned to the top of a specified row and left side of a specified column plus offsets from that location. Move And Size placement links the control size to the width and height of cells between the Top Left and Bottom Right reference cells.

A picture added to a Microsoft Excel document is also treated as a form control within ExcelRT. While a picture uses the same type of placement and positioning data as other form controls, the picture itself is stored separately from the control.

A Picture control refers to a Resource ID that holds the picture. The same picture can be used for multiple controls and scaled as needed to reduce size and optimize performance. Pictures are discussed later in this chapter.

Double-click on the **Control Edit** tool in the Toolbar to present the Form Controls dialog to Add, Edit, Delete or view any form controls in the project.



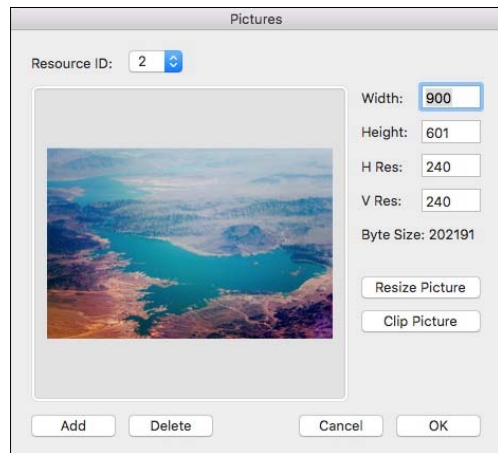
Controls are drawn on a sheet based on the top down order of controls in the Form Controls dialog. Use the **Up** and **Down** arrow button in the dialog to reorder controls if needed.

Pictures

The **Pictures** button presents a dialog to add, delete or modify pictures. Notice how each picture has a fixed Resource ID that a form control can reference.

The Width and Height of a picture determines how large it is on the screen for a form control with Free Floating or Move placement.

The Byte Size of a picture is very important since it affects the size and performance of every ExcelRT file created by your application. To resize or clip a picture, first change the Width and Height fields, then click the **Resize Picture** or **Clip Picture** buttons.



Cell Control



Unlike Form Controls that have a Microsoft Excel equivalent, Cell Controls are unique to ExcelRT.

Each cell control turns the cell itself into a control. Similar types of controls are supported including push buttons, checkboxes, radio buttons and popup menus.

Cell controls are often easier for a developer to manage since the cell itself determines the position and visibility.

When a user clicks a cell control, that action often triggers a script to run. A typical script command is shown in blue for each control type.

When a cell control is type Picture, a button is visible to present the Cell Control Pictures dialog. Pictures for cell controls are not read or written in the ExcelRT file itself, but rather loaded into memory at runtime using a Script command. The same picture can be used in many cells.

Cell control pictures typically come from an image file in the Plugins folder or from the Internet. These images often provided by the workbook user rather than the designer.

There is also a Script command to create a cell control picture from a form control picture that is stored in the ExcelRT file.

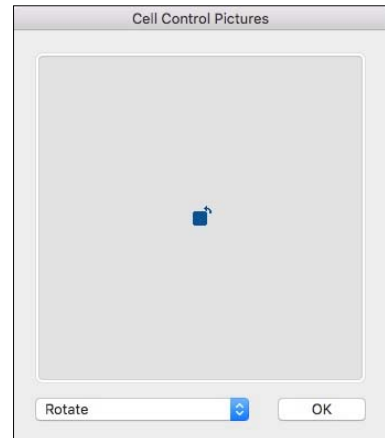
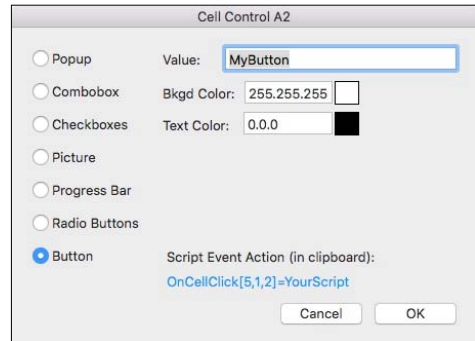
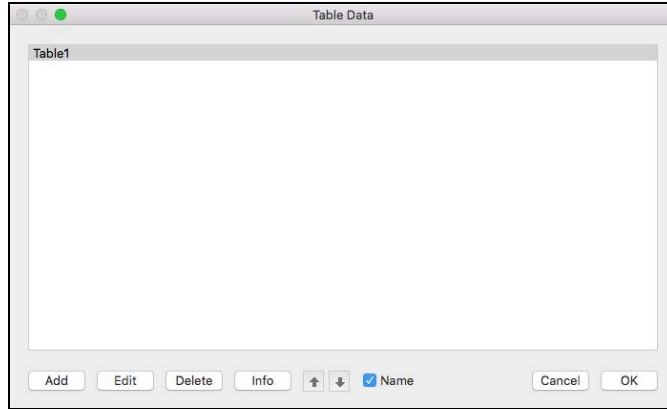


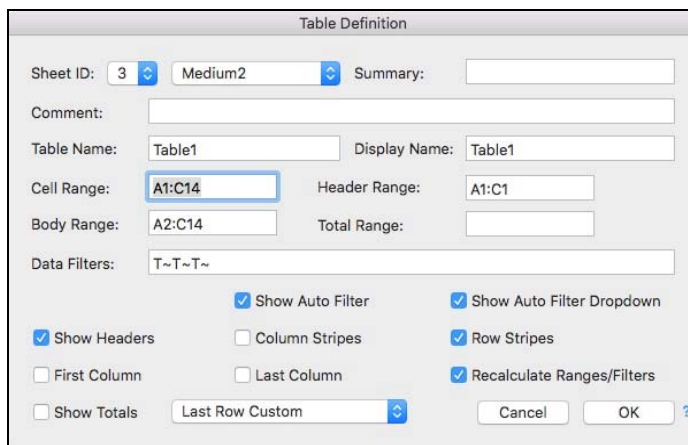
Table Add Delete

T Click to present the Table Data dialog to add, edit or delete tables within the project. Tables overlay extra properties on a range of cells.



To add or edit the properties of a specific table, present the Table Definition dialog. Select the sheet where the table resides. Select a style and give it a name.

Select the Cell Range of the table. For most tables, other ranges can be calculated for you automatically based on other options you select. Set checkboxes to determine if the table has headers or other characteristics. Use the popup menu to determine if the last row or column has a specific or custom formula applied.



Cell Borders



Click to present the Cell Borders dialog. After assigning borders to the selected cell, those cell borders can be copied and pasted to other cells using the **Copy** and **Paste** button.

Cell Borders

From Column: A From Row: 3 To Column: A To Row: 3

Border Type	Direction	Line Style	Color	Preview
Continuous	Bottom	Hairline	251.1.6	[Red]
Continuous	Left	Hairline	251.1.6	[Red]
Continuous	Right	Hairline	251.1.6	[Red]
Continuous	Top	Hairline	251.1.6	[Red]
None	Bottom	Hairline	0.0.0	[Black]
None	Bottom	Hairline	0.0.0	[Black]
None	Bottom	Hairline	0.0.0	[Black]
None	Bottom	Hairline	0.0.0	[Black]

Clear Copy Paste Cancel OK

Cell Validation



Click to present the Cell Validation dialog. Assign validation rules, input messages and alert messages and actions. After assigning validation rules to the selected cell, those rules can be copied and pasted to other cells using the **Copy** and **Paste** button.

Cell Validation

From Column: A Row: 3 To Column: A Row: 3

Allow: Decimal With Data: greater than

Minimum: 0 ☐ Ignore Blank

☐ Show input message when cell is selected

☒ Show error alert after invalid data is entered Stop


Error Alert

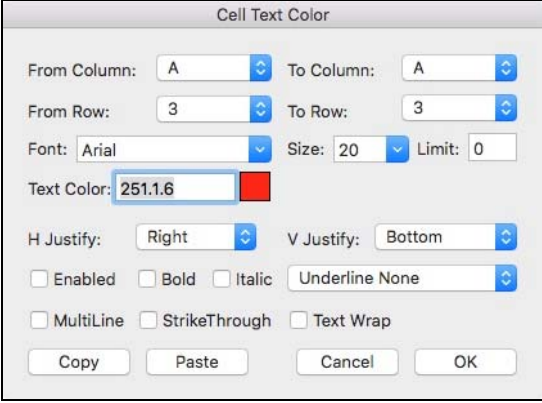
Title: [Text Box]

Message: [Text Box]

Clear Copy Paste Cancel OK


Cell Text Color

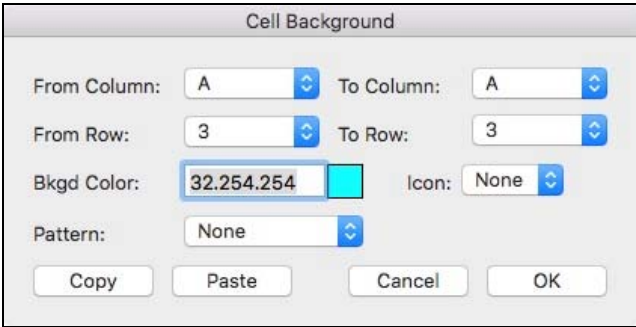
 Click to present the Cell Text Color dialog. Change font, size, color, style and other text properties of a cell. The From Column and Row and To Column and Row fields show the selected cell. Text properties can be applied to a range of cells.



The 'Cell Text Color' dialog box is shown. It has a title bar 'Cell Text Color'. The fields are: 'From Column:' A, 'To Column:' A, 'From Row:' 3, 'To Row:' 3, 'Font:' Arial, 'Size:' 20, 'Limit:' 0, 'Text Color:' 251.1.6 (with a red color swatch), 'H Justify:' Right, 'V Justify:' Bottom, 'Enabled' (unchecked), 'Bold' (unchecked), 'Italic' (unchecked), 'Underline None' (selected), 'MultiLine' (unchecked), 'StrikeThrough' (unchecked), 'Text Wrap' (unchecked). At the bottom are buttons: 'Copy', 'Paste', 'Cancel', and 'OK'.

Cell Background

 Click to present the Cell Background dialog. Use this dialog to apply a background color, pattern or icon to the selected cell or a range of cells.



The 'Cell Background' dialog box is shown. It has a title bar 'Cell Background'. The fields are: 'From Column:' A, 'To Column:' A, 'From Row:' 3, 'To Row:' 3, 'Bkgd Color:' 32.254.254 (with a cyan color swatch), 'Icon:' None, 'Pattern:' None. At the bottom are buttons: 'Copy', 'Paste', 'Cancel', and 'OK'.

Cell Copy & Paste



Click to copy a range of cell properties from the current sheet. Copied data can be pasted on other sheets or workbooks.

Only checked properties are copied. The Value is the actual data entered into the cell. A cell Formula or Dependency can be copied, but may need to be adjusted once copied into its new location.

The **Build Dependency** command from the **File** menu to recalculates the dependencies of all cells in the workbook.



Click to paste a range of copied cell properties into the current sheet.

Cell Range Copy

From Column: A To Column: E

From Row: 1 To Row: 10

- ☒ Value
- ☒ Comment
- ☒ Formula
- ☒ Dependency
- ☒ Choices
- ☒ Format
- ☒ Background-Color
- ☒ Text-Color
- ☒ Icon
- ☒ Text-Style
- ☒ Pattern
- ☒ Bar-Properties
- ☒ Validation
- ☒ Cell-Type
- ☒ Merge-Area
- ☒ Borders
- ☒ Text-Font
- ☒ Text-Size

Check All Uncheck All Cancel OK

Cell Range Paste

From Column: A To Column: E

From Row: 1 To Row: 10

Cancel OK

Paste Cell Range

In addition to clipboard data, other types of data can be pasted into a range of cells.

Set the Value radio button and enter a specific value to be pasted into each cell of the range. For testing purpose, create random integer, float or string data to be pasted into each cell.

The Repeat Row Adjusted Formula option can apply the same formula to a range of cells and adjust the row references. In the example shown, on row 3 the pasted formula is adjusted to $=A3*B3+2-(A22-2)$

Cell Range Paste

From Column: C To Column: C

From Row: 2 To Row: 6

Data Source

☐ Clipboard ☒ Value: Test Data

☐ Random Integer From: 1 to 999

☐ Random Number From: 1.1 to 999.999


☐ Random Text From: a to az1A29

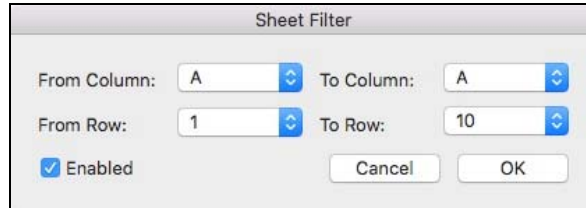
☒ Repeat Row Adjusted Formula

$=A2*B2+2-(A22-2)$

Cancel OK

Sheet Filter

 Click to enable or disable a Filter on the current sheet. A filter is used to sort and organize data and is similar to a simplified table.



The 'Sheet Filter' dialog box has a title bar 'Sheet Filter'. It contains four dropdown menus: 'From Column:' set to 'A', 'To Column:' set to 'A', 'From Row:' set to '1', and 'To Row:' set to '10'. Below these is a checkbox labeled 'Enabled' which is checked. At the bottom right are 'Cancel' and 'OK' buttons.

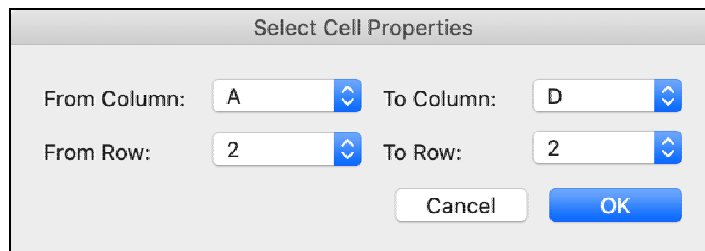
Merged Cells

ExcelRT supports the Excel concept of merged cells. Multiple cells can be treated as one combined cell for user editing and display.

	A	B	C	D
1	A1	B1	C1	
2	Merged Across A2:C2			
3				

To implement this in ExcelRT Builder, put the cell range into the Merge Area field of every cell in that merged cell range. If the area consists of A2:C2, then put this text in the Merge Area field of cells A1, B1 and C1. If all cells in the range don't include the area, ExcelRT will not display the merged cell as expected.

When clicking in a merged cell range with the Cell Properties tool, the top left cell is selected within the Cell Properties dialog regardless of which cell you click on in that range. To edit the properties of a specific cell, click to the left of the Column A header to present the Select Cell Properties dialog. Select a specific column and row, then click **OK**.



The 'Select Cell Properties' dialog box has a title bar 'Select Cell Properties'. It contains four dropdown menus: 'From Column:' set to 'A', 'To Column:' set to 'D', 'From Row:' set to '2', and 'To Row:' set to '2'. At the bottom right are 'Cancel' and 'OK' buttons.

The Value of the top left cell is what actually gets displayed on the screen or edited by the user. The Values of all other cells in that range are ignored.

HTML Control

The Control Add, Delete, Edit and Move tools can be used to define one or more HTML Viewer controls on each sheet of the workbook. The displayed content may be offline from the Plugin folder or live from the Internet.

An HTML Control can display a folder of HTML related files from the Plugins folder. The HTML content may consist of hundreds of files and nested folders control HTML, CSS and Javascript files.

Form Control

Control Type: HTML Name: Slide

URL or Plugin: Landscapes

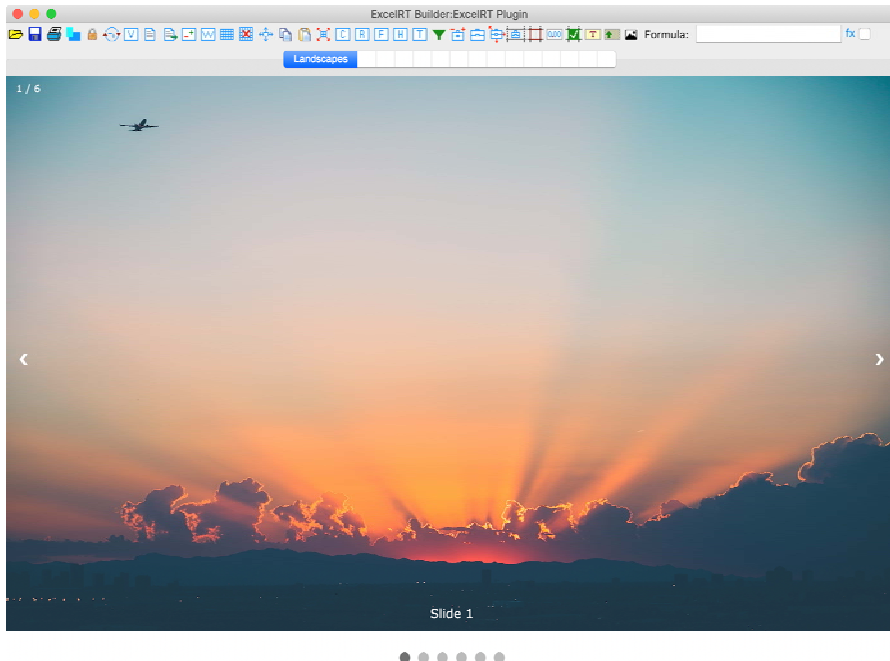
Shape Top: 0 Left: 0 Width: 800 Height: 700

Top Left: Row: 1 Column: 1 Top Offset: 0 Left Offset: 0

Bottom Right: Row: 1 Column: 1 Bottom Offset: 0 Right Offset: 0

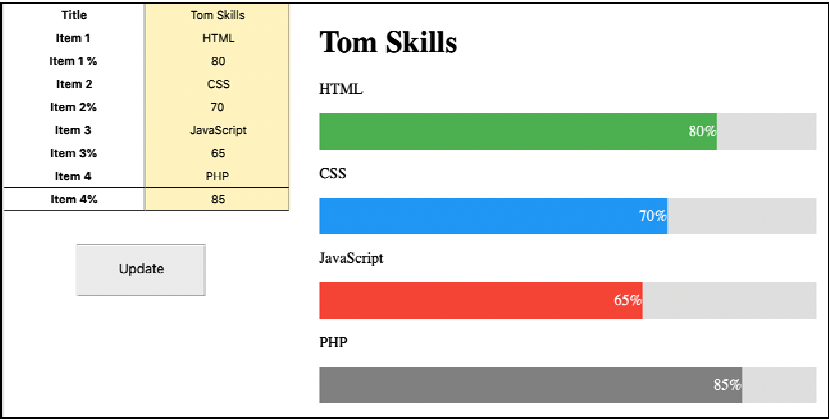
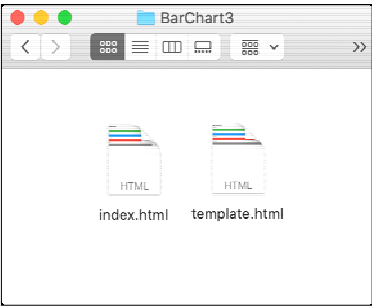
Placement: Free Floating Visible

Cancel OK



HTML based technologies can be an integral part of the ExcelRT workbook. Scripting commands can show, hide or change the content displayed within HTML Viewer controls. Workbook data can drive the HTML content or calculated Javascript results can be returned to the workbook.

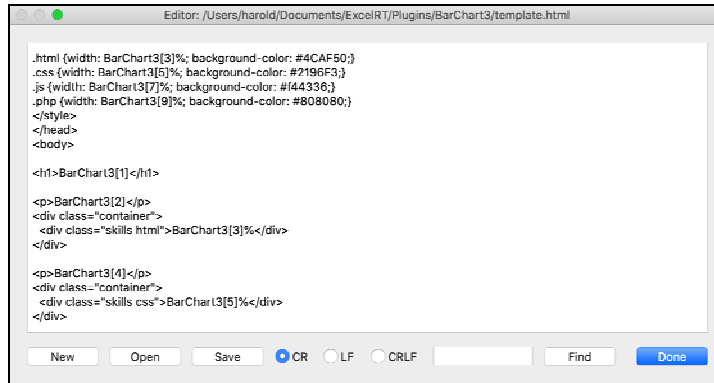
Below is an example of a BarChart driven by workbook data. The shaded cells are user editable data that is read into an array with a script command, then sent to a file in the Plugins folder to be displayed in the HTML viewer as a dynamic bar chart.



The HTML Viewer control is displaying the index.html file. When a user clicks the **Update** button, this script runs.

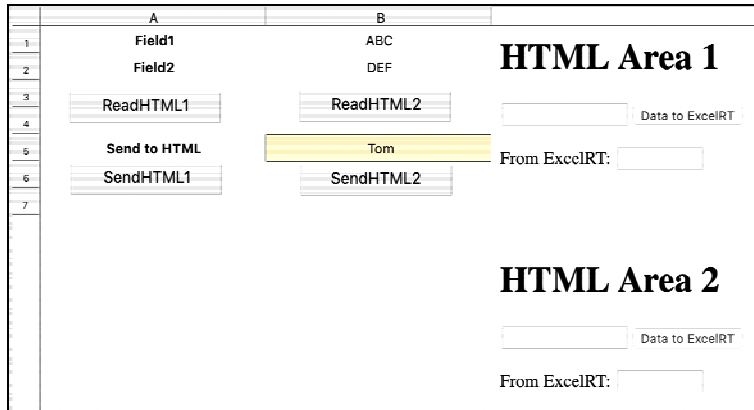
```
UpdateBarChart3Btn=ArrayLoad(BarChart3,B1:B9,ALL)|PluginTaggedTemplate(template.html,index.html,BarChart3,BarChart3)
```

Data from cells B1:B9 is read into an array named BarChart3. The PluginTaggedTemplate command generated index.html from template.html by replacing array tags with data. For example, the value of B1 is used as a header in the chart. That value is read into the first element of the array, that used to replace BarChart3[1] in the template.html file before generating index.html.



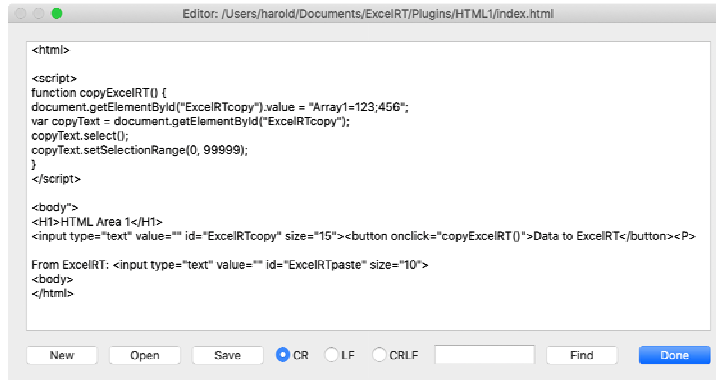
To recap, start with HTML content that can be displayed in an HTML Viewer control. Clone the index.html file (or any text file) to a template file and added array tags. Now add a few lines of text to define the array values and call the PluginTaggedTemplate command. The script can be called from a button or an event like switching sheets.

Data can be copied and pasted between the workbook and HTML environments. This sheet contains two HTML controls illustrated as HTML Area 1 and 2. When the Data to ExcelRT button is clicked, the value of the ExcelRTcopy element is set.



When the user clicks the ReadHTML1 button, the follow script command run and copy data from the ExcelRTcopy element with the HTML environment to cells B1 and B2 within the workbook.

```
ReadHtml1Btn=HtmlViewerCopy(Array1,StatusVar)|ArraySave(Array1,B1:B2)
```



The OnHTMLViewer event, can further automate this process as illustrated with this script code. The format of the ExcelRTcopy element is array name, equal plus each array element value separated by semicolon.

```
OnHtmlViewer[Array1]=HtmlViewerCopy(Array1,Stat)|ArraySave(Array1,B1:B2)|Redraw()
```

This HtmlViewerCopy command uses clipboard copy in the native OS where the ExcelRT workbook is running. For security, modern browsers require that the HTML element being copied is visible and entered by the user or set by a Javascript action triggers by a user button click. The size of the HTML element can be minimized.

The HtmlViewerPaste command assigns data from the workbook to an element in the HTML environment. Despite the name, this command is not actually using the clipboard. The HTML viewer is directly changing the value of an element within the HTML page.

Assume the user types data into B5 and clicks the SendHTML1 button. These script commands run and modify the value of the ExcelRTpaste element in the HTML.

```
SendHTML1=VarFromCell(Data,B5)|HtmlViewerPaste(HTML1,Data,StatusVar)
```

An HTML control can get its source from a script variable. In the Form Control dialog, enter Var=VarName into the URL or Plugin field as illustrated below for a variable named MyHtml. To demonstrate, use the CsvToHtml command to build the HTML page from a loaded CSV file.

```
Var=MyHtml
```

An HTML control uses local browser technology from the computer or device it is running on. Some HTML constructs are rendered different or not at all on some browsers or devices.

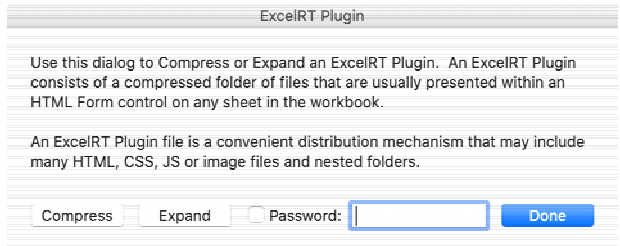
ExcelRT Plugin

For convenience and distribution, ExcelRT provides a command to compress a folder of files nested within the Plugins folder into a single Plugin file. To use those files, an expand command restores the original folder of files.

Use the ExcelRT Plugins command on the **Files** menu to present the ExcelRT Plugins dialog. Use the **Compress** button to create a Plugin from a selected folder in the Plugins folder. The **Expand** button restores a folder from a selected Plugin file.

See the PluginCompress and PlugExpand script commands to compress and expand plugins within a running ExcelRT workbook.

Although Plugins are typically used to distribute HTML content, they can be used for other purposes.



Plugins can be encrypted with a password. Only users with the encryption password can expand the Plugin to see or change the original source files within the Plugin.

A password-encrypted plugin can be used within an ExcelRT workbook without expanding the plugin to its source files or without knowing the encryption password. The HTML control on ExcelRT sheet refers to the plugin name XXX.excelrt_plugin.

If an ExcelRT plugin stored in the Plugins folder is not password encrypted, it will automatically expand into the Plugins folder on first use of the workbook. If the plugin is password encrypted, it runs within a virtual environment. If the plugin is very large, maintaining this virtual environment may slow down the workbook.

If the PluginTaggedTemplate feature is used with a password-encrypted plugin, the FolderName parameter is the actual plugin name as illustrated here.

```
PluginTaggedTemplate(template.html,index.html,BarChar,BarChart.excelrt_plugin)
```

Using an ExcelRT Plugin within an ExcelRT Cloud account has special significance. For example, a single copy of a plugin file can be shared across all user accounts and provide a private virtual instance to each specific account as needed. Refer to the ExcelRT Plugin section of the ExcelRT Cloud chapter.

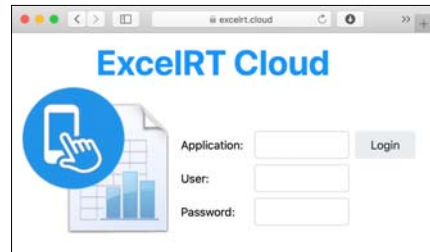
Chapter

8

ExcelRT Cloud

ExcelRT Cloud allows ExcelRT workbook applications to be run from a web browser on virtually any computer or device. The ExcelRT file is developed and tested on a Mac or Windows computer, then uploaded into a Vendor account in ExcelRT Cloud.

An ExcelRT Cloud based application runs in a browser with similar capabilities to an App running on a local computer. Users can navigate between workbooks, between sheets of a workbook, enter data, import or export files, present dialogs, share data between workbooks or interact with Internet websites.



ExcelRT Cloud is a monthly service that provides a developer with a fully managed ExcelRT Cloud server. The Vendor and Users log in from a web browser.

Base on the account size, the ExcelRT Cloud server can support multiple applications and many users.

ExcelRT Cloud can optionally be linked to a Safe Activation account to manage an automated order, Serial Number delivery and account creation process. An ExcelRT Cloud account can be linked to a Cloud Sharing account to share files with other Cloud or Desktop Apps.

Define Apps

A developer can log into their Vendor account to define Apps and Users. To log into a Vendor account, type Vendor in the application field, your 6-digit ExcelRT Cloud Vendor ID in the User field and your password.

The main screen consists of a scrolling App and User list. Each user can be assigned to one or more Apps.

The screenshot shows the ExcelRT Vendor interface. At the top, it says "ExcelRT: 3.1 Apps: 3/4 Users: 2/25 Max MB: 4". There are buttons for "Log Out", "Password", "Restore", "Goto Control Panel", "Settings", "Feedback", "Plugins", "Stats", "Add", "Edit", "Delete", and "Confirm Delete". Below this is a table of apps:

App ID	App Name	Enabled	AutoSave	Master	Size KB	Password	Support	Purchase	Feedback	Settings
2009011	BridgeScore	Yes	Yes	BridgeScoreRank.ert	169	Test	Yes	Yes	Yes	No
2009012	TravelCalc	Yes	Yes	TravelCalc.ert	39	Test	No	No	No	No
2009013	Quote	Yes	Yes	Quote.ert	192	Test	No	No	No	No

Below the app list are buttons for "Orders", "Process Orders", "Batch", "Search", "Add", "Edit", "Delete", "Confirm Delete", "Plugins", "Stats", and "Profile". At the bottom is a table of users:

User ID	Password	Contact	Notes	Email	Notify	Licenses
2009011	Pswd1	John Doe				BridgeScore,TravelCalc,Quote
2009012	Pswd2	Jane Doe				BridgeScore,TravelCalc

To define an App, click the **Add** button above the App List to enter data and upload a completed ExcelRT file. The minimum data required to define an App consists of the App Name, an uploaded ERT file, that file name in the Master Workbook field and the Open Password.

The screenshot shows the "Add App" form. It contains the following fields and options:

- App ID: 2009011
- ☒ Enabled ☒ AutoSave
- Login Title: [text box]
- App Name: BridgeScore
- Login Image: [text box]
- Workbook: BridgeScoreRank.ert
- Max Sessions: 2 (dropdown) Sheet Select Width: 1104
- Open Password: Test
- ☒ File Manager ☐ Click Beep ☒ Save Button
- Support URL: https://www.excelsoftware.com/sup
- Purchase URL: https://www.excelsoftware.com/buy
- Feedback: Type Feedback for App Developer
- Notify: Type text to Notify user.
- ☒ Cloud Sharing ☐ Serial Number Lookup
- Cloud Vendor ID: [text box]
- Default Accounts [button]
- Cancel [button] OK [button]

Define Users

To define a User, click the **Add** button above the User List. Each user has a unique Contact name they must enter into the User field when logging into a specific App. Each user account is given a default password that can be changed from the Vendor account or by the user after they log into their account.

User ID: 2009011

Password:

Contact:

☐ Alert

Notes:

Email:

Notify:

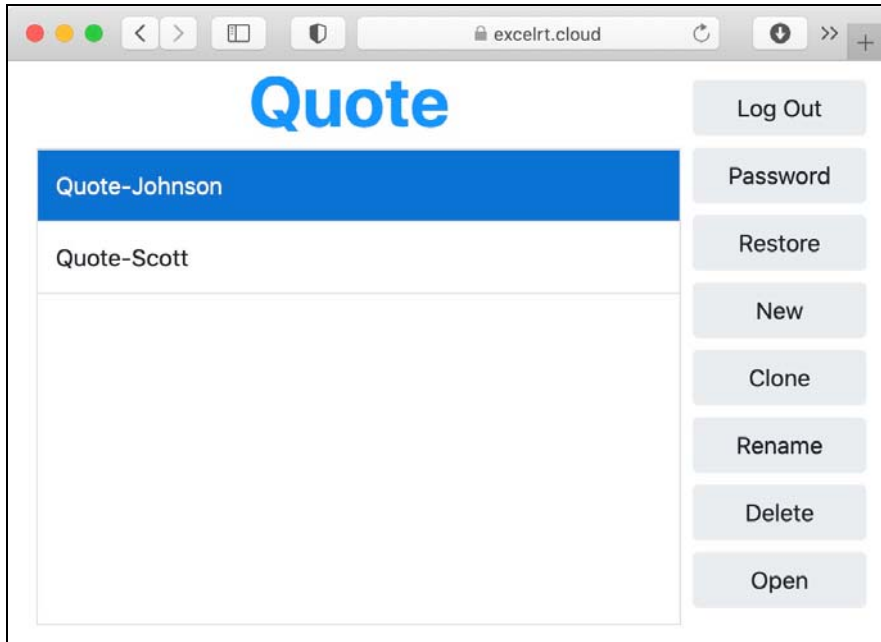
App Name	License	
BridgeScore	Yes	Yes
		No
TravelCalc	Yes	Cancel
Quote	Yes	OK

The User dialog shows a list of available App names that can be assigned to that user by selecting the App name and clicking the **Yes** or **No** button.

User Experience

A user logs into a specific App within their account, by entering the App name, their User name and password. When the App is configured, the vendor determines if the user logs directly into the ExcelRT workbook or presents a File Manager screen as illustrated here.

The user can create, rename, clone or delete a list of workbook files using buttons in the File Manager screen. To open a specific workbook file, select it and click the **Open** button.



ExcelRT Cloud maintains a folder for each App and User combination where workbook files are stored. Those files are backed up daily so the user can restore their account if needed or change their password using buttons at the top right edge of the window.

Customer	
Name	John Doe
Billing Address	124 Main
City State ZIP	Henderson NV 89015
Phone	702-445-7645
Email	john@gmail.com

Project	
Project Name	House Remodel
Project Address	124 Main
Description	Remodel Sun Room Plus Bedrooms 1 and 2

Quote	
Number	12345
Date	June 15, 2020

Clear

Once a user is in the workbook, it functions almost exactly the same as it does when running as a desktop App. Click on a cell and type to enter data. Click buttons to present dialogs for data entry. Select a sheet name at the top to switch to a different workbook sheet.

When editing data in a workbook, click the **Save** button at the top left to save it to disk. The App can optionally be configured to Auto-Save when closing the workbook. One difference between a Cloud and Desktop App is that if you leave the computer, after a period of time your session will automatically close. Likewise, if you loose Internet access, your session will close.

Custom File Manager

To present the File Manager screen for an App, set the File Manager checkbox when defining the App. If unchecked, the App opens directly into the workbook. The App is always presented within one browser screen. Modal dialogs may be presented within that window.

To include a **Support** button in the File Manager screen, enter a URL in the Support field when defining the App. That button takes the user to a support page on your website in a separate browser panel.

To include a **Purchase** button, enter a URL in the Purchase field when defining the App. That button takes the user to a Purchase page on your website in a separate browser panel where they can buy Serial Numbers for other products you sell.

The **Feedback** button presents a dialog that collects developer-configured information from the user and logs it to the Vendor account. The vendor can review time-stamped feedback logs from users running the various Apps they offer.

The **Cloud** button presents a dialog linked to a Cloud Sharing account. The user can upload and download files or switch to a different Cloud Sharing account.

The **Setting** button presents a dialog that collects developer-configured data from the user. That data is stored in a Settings.data file in the user's application folder. The data is easily accessible to any workbook on open using a script command. Common information like Name, Company, Address or Phone need only be entered or modified once by the user and then included in each workbook.

Settings

To configure a Settings button for an App, select the App and click the **Settings** button in the ExcelRT Vendor account. Use the Setting Template dialog to define the labels and field type of each data field collected from the user.

Field Name	Field Type	Comma Separated Choices
First Name	Edit	
Last Name	Edit	
Company	Edit	
Phone	Edit	
Customer Type	Radio	Company,Individual
<div>AddEditDeleteCancelDone</div>		

In this example, the First Name, Last Name, Company and Phone fields are text edit fields. The Customer Type field is a set of Radio buttons with names Reseller, Affiliate, Company and Individual. Other supported field types include Section, Checkbox, Popup and ComboBox.

Field Name:

Customer Type

Field Type:

Radio

Choice List:

Company,Individual

Cancel

OK

To add a field to the Settings dialog, click the Add button and complete the presented dialog.

When the user clicks the **Settings** button within your App, a dialog collects data from the user and stores it in a file that your application can easily retrieve.

First Name

John

Last Name

Doe

Company

ABC

Phone

702-445-7645

Customer Type

Company

Individual

Cancel

OK

Feedback

To collect feedback from users, enter a label in the Feedback field when defining the App in the ExcelRT Vendor account.

An App user can provide feedback to the developer by clicking the **Feedback** button in the File Manager window.

From the ExcelRT Vendor account, click the **Feedback** button to present the Feedback Reader dialog. Each time stamped feedback file contains the Date, App and User. Click the **Delete** button to delete the file begin viewed. To edit the file, simply type typing and then click the **Save** button to save your changes.

Give feedback to App developer

An App user can type feedback here.

The developer can read that feedback from the Vendor account. Each feedback enter records the User, Date and App name.

CancelOK

20210904_13_44_51	20210904_13_46_15
20210904_13_46_15	<p>Date: 20210904 App: Quote User: 2009011</p> <p>Please provide a button to email a PDF of the project summary to my customer.</p>
<div><button>Delete</button><button>Save</button><button>Goto User</button><button>Done</button></div>	

When viewing feedback from a specific user, click the **Goto User** button to present the User dialog for the person providing the feedback. Type text into the Notify field to notify. Perhaps a bug was field or you need to request additional information.

A user with a Notify message is highlighted within the Vendor account so that notification can later be cleared.

Cloud Sharing

To support Cloud Sharing in your Apps, set options in the App Edit dialog within an ExcelRT Vendor account.

☒ Cloud Sharing

☒ Serial Number Lookup

Cloud Vendor ID:

Default Accounts

Set the Cloud Sharing checkbox to include the Cloud button in the file manager screen. To allow lookup of Cloud Sharing credentials based on a Serial Number received during the purchase process, set the Serial Number Lookup checkbox and enter your Cloud Vendor ID. Use the **Default Accounts** button to provide one or more Cloud Sharing accounts of your own for use by customers.

Refer to the Cloud Sharing User Guide for a complete description of the setup process and user experience.

When a user clicks the **Cloud** button, the Cloud Sharing dialog is presented. The user can Upload, Download or Delete files from their account. The user can assign or switch between multiple Cloud Sharing accounts.

An ExcelRT workbook can present the Cloud Viewer dialog with a CloudViewer script command.

This dialog is a portal to specified file types in a Cloud Sharing account. Select My Account radio button to upload from or download to the Plugin folder of the ExcelRT Cloud account.

Select My Device to upload from or download to the local disk of the device. The downloaded file is stored in the Downloads folder of a computer. On an iPad, the file is retrieved and displayed allowing the user to touch the image for additional sharing options.

Cloud Sharing: MyCloud [Write]

Johnson Quote 20210902.ert	Setup Upload Download Delete Done
Scott Quote 20210901.ert	
Thomas Quote 20211005.ert	

My Cloud

Johnson Quote 20210902.ert	<input checked="" type="radio"/> My Account	Upload Download Delete Done
Scott Quote 20210901.ert	<input type="radio"/> My Device	
Thomas Quote 20211005.ert		

User Notify and Alert

The ExcelRT Vendor account has several features to help you manage user accounts. For example, assume that you manage user accounts and notice that a user's credit card has expired.

To present a notification message when a user logs into their account, select that user row and click the **Edit** button. Type info into the Notify field as shown below then click **OK**. The Notify column of that row is highlight with **Yes** in red.

Each time the user logs in to use your application, they see the Notify message in a dialog. Once the credit card info has been updated, clear the Notify message from the User edit dialog.

The screenshot shows a 'User Edit' dialog box. At the top, there are fields for 'User ID: 2009011' and 'Password: Pswd1'. Below these is a 'Contact' field with 'John Doe' and an 'Alert' checkbox which is checked. There are also empty fields for 'Notes:', 'Email:', and 'Notify:'. The 'Notify' field contains the text 'Credit card expired. Update info in portal.' Below the 'Notify' field is a table with three columns: 'App Name', 'License', and a third column with buttons. The table has three rows: 'BridgeScore' with 'Yes' and a 'No' button; 'TravelCalc' with 'Yes' and a 'Cancel' button; and 'Quanta' with 'Yes' and an 'OK' button.

App Name	License	
BridgeScore	Yes	No
TravelCalc	Yes	Cancel
Quanta	Yes	OK

The Alert checkbox in the User dialog is another way to mark a customer for special attention. Notice the User ID of that row is highlighted in red. When using an automated Paypal Subscription button and Serial Number in Safe Activation, an Alert can automatically be set when a subscription is cancelled by the user or is allowed to expire.

If a user wants to permanently cancel their subscription, present the User dialog, select the application and click the **No** button. Be careful, this action deletes the user account and all application related files created by that user.

Batch User

Click the **Batch** button in the ExcelRT Vendor account to manipulate a batch of user accounts. The **Import** and **Export** button allows user accounts to be imported or exported from CSV (comma separated value) formatted text

Add Users:

This dialog will close after adding, deleting or importing users. The process may take over a minute to complete.

Number of Users:

Apps:

Add

Delete Users:

From User ID:

To User ID:

Delete

Export CSV:

Export

Import CSV:

Select

Import

Contact,Password,Email,Notes,Licenses

Licenses=AppName1|AppName2

Done

The **Export** button downloads a CSV formatted text. It gives you the option to a new tab in the browser with that data. Data can be copied and pasted into another application like Microsoft Excel. Notice the first line consists of a comma separated header row. Each additional row contains the information for one user account.

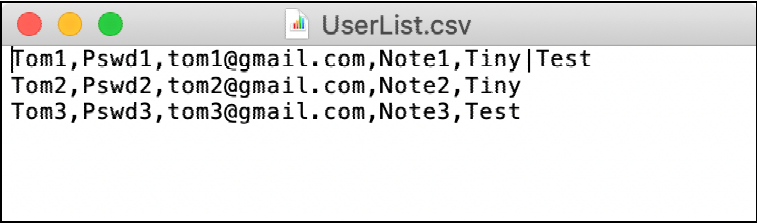
excelrt.cloud

Vendorhttps://excelrt.cloud/print/Vendor200320....

UserID,Password,Contact,Email,Notes,Notify,Licenses,Alert
2003201,Pswd1,John Doe,,,Bug fixed,BridgeScoreRank|My Great App,No
2003202,Pswd2,Jane Doe,,,BridgeScoreRank|TravelCalc,No
2003203,Pswd1,Tom1,tom1@gmail.com,Note1,,,No
2003204,Pswd2,Tom2,tom2@gmail.com,Note2,,,No
2003205,Pswd3,Tom3,tom3@gmail.com,Note3,,,No

The **Import** button accepts CSV formatted data from a text file into the ExcelRT Vendor account. Notice there is no header row, just one line of comma-separated data for each user account that you want to add.

The list of fields for each account include account, password, email, notes and Apps where the Apps field can be multiple App names separated by |.



```
Tom1,Pswd1,tom1@gmail.com,Note1,Tiny|Test
Tom2,Pswd2,tom2@gmail.com,Note2,Tiny
Tom3,Pswd3,tom3@gmail.com,Note3,Test
```

User accounts are typically added manually by a developer as customers subscribe to Apps they provide or through an automated purchase process using a Serial Number driven approach with the Safe Activation service.

Use the **Add** button to create a batch of accounts that will be distributed to existing users or through some other type of shopping cart or automated purchase process. With this approach, the account name is a unique number (same as the User ID) created in sequential order plus an assigned random number password. After creating the batch of user accounts, export the data for distribution to customers. After logging into their account, each user can change their password.

Use the **Delete** button to delete a range of user accounts based on the User ID.

Plugin Folder

Information in this section applies to more advanced workbook applications that use script commands.

Every user account has a Plugin folder. If the user account is assigned to multiple Apps, they all share the same Plugin folder. Apps can read and write files to the Plugin folder using simple script commands. Data can also be shared between different Apps using the Plugin folder.

The Plugin folder provides disk storage for images or csv data files. For example, assume your App includes a button that downloads a current data file from your website to the Plugin folder, then uses that data to populate fields within the workbook. The data only needs to be downloaded once and can then be shared across all workbook files in that user account.

Some apps include additional image or data files that only need to be stored once in the Plugin folder for each user account. This can reduce the file size of a workbook so it opens and saves faster. This also provides a mechanism where the developer can provide updated information or user specific information as need.

A developer uploads Plugin files using the Upload section of the App Edit screen.

Uploaded files for an App can be copied into the Plugin folder for each newly created user account. Select the App from the Vendor screen and click the **Plugins** button to present the Plugin Files dialog. Select a file that should be copied to each Plugin folder, then click the **Copy Yes** button to mark that row as Yes.

Filename	Copy to New User Plugin Folder
img1.jpg	No
img2.jpg	No

Copy To User ID

Renamed To:

Delete

Copy To All Users

Copy Yes

Copy No

Shared

Replace Master for Assigned Users

Done

Files that need to be read, but never changed by individual user accounts should always be marked as Shared. When a file is marked as Shared, it can be accessed with script commands as if it was stored in the local Plugin folder for that account. Storage space is dramatically reduced since only one copy of the file exists on the server. That also makes an update to the file much easier for the vendor.

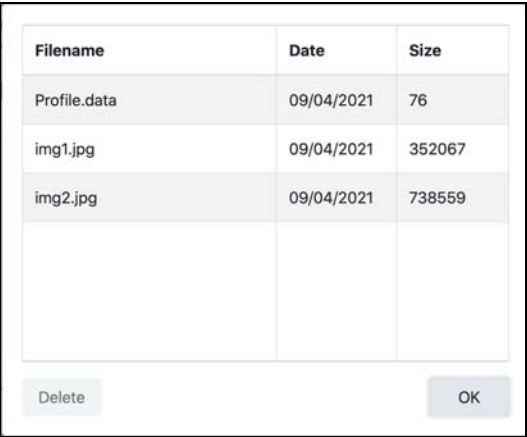
After marking a file as Shared (or not Shared), the vendor must **Stop** and **Start** the ExcelRT Cloud server from the ExcelRT Control Panel for that change to take affect in user accounts.

To copy a selected file to the Plugin folder for a specific user account, enter the User ID in the edit field and click the **Copy To User ID** button. If you want that file renamed after it is copied into the Plugin folder, enter the new name in the Renamed To field before clicking the button.

To delete a file uploaded when the App was originally defined, select it and click the **Delete** button. To copy a selected file into every user account to which that App is assigned, click the **Copy To All Users** button.

App users don't need to understand the concept of the Plugin folder nor do they have a direct mechanism to view or alter the contents of that folder. An App can provide buttons and script commands to move files to and from the Plugin folder.

When debugging an App, it can be useful for a developer to see the contents of the Plugin folder for a specific user account. Select the user in the User List of the Vendor account and click the **Plugins** button to present the Plugin List dialog.



The screenshot shows a dialog box titled 'Plugin List' with a table containing three columns: 'Filename', 'Date', and 'Size'. The table lists three files: 'Profile.data', 'img1.jpg', and 'img2.jpg', all dated '09/04/2021'. Below the table are two buttons: 'Delete' and 'OK'.

Filename	Date	Size
Profile.data	09/04/2021	76
img1.jpg	09/04/2021	352067
img2.jpg	09/04/2021	738559

Delete OK

Each file in the Plugin folder of that user account is shown with the modification date and size. To delete a file, select it and click the **Delete** button.

Update an App

Most Apps consist of an ERT file (master workbook) uploaded by the developer when the App is defined. The developer may later fix bugs or add enhancements to the master workbook and want to upgrade existing users to the new workbook.

If an App uses the File Manager, the **New** button copies the master workbook to the App folder for that user account and names it. After making improvements to the master workbook, the developer can simply upload the new ERT using the Add Edit dialog. If the user clicks **New** in the File Manager, a copy of the new master workbook is created. Existing named workbooks are unaffected.

If changes to the new master workbook are such that some users may want to upgrade and other may not, then create a new App with a slightly different name. For example, if the first app was called TravelCalc, perhaps the next major release is called TravelCalc2.

The developer must assign the new App to each user account that needs to run it. To log in, the user will type TravelCalc or TravelCalc2 in the App field. This approach can be used to provide a Beta version of a new App to a limited number of users.

If the App does not use the File Manager, the master workbook is copied into each newly created user account. Each time a user runs the App, the same file is opened and saved. Uploading a new master workbook has no affect on existing users.

To replace the master workbook in existing user accounts when no File Manager is used, select the App from the Vendor screen and click the **Plugins** button. Now click the **Replace Master for Assigned Users** button.

Custom Login

By default, a generic Login screen is presented to all users regardless of the App they purchased. The URL for the Login screen takes the form shown below.

`https://excelrt.cloud:9005`



The number at the end of the URL is a port number that will be unique for each developer's ExcelRT Cloud account.

The Title, Image and default App name on the Login screen can be customized by defining the Login Title and Login Image fields in the App Edit dialog.

Login Title:	<input type="text" value="My Great App"/>	?
Login Image:	<input type="text" value="AppIcon.png"/>	

Upload an image with transparent background and enter the name of that image file in the Login Image field such as AppIcon.png. You can upload a different image for each App. In this example, the Application name field is defaulted to YourApp and an App specific image and title is shown in the Login screen.

To apply the Login screen customization, provide the user with a custom URL of the form below. Use your Port number, your Vendor number and your App Name.

`https://excelrt.cloud:9005/?vendor=191201&app=YourApp`

For multiple Apps, you can supply the Login Title and Image field in just one app. Change the URL to use the logo field assigned to that App name as shown here.

`https://excelrt.cloud:9005/?vendor=191201&logo=YourApp`

You can share the same Title and Image on all of your apps, but still default the App name field so the user can avoid typing it by using three parameters as shown here.

`https://excelrt.cloud:9005/?vendor=191201&logo=YourApp&app=AppTwo`

Browser & Device User Experience

ExcelRT Cloud supports Mac, Windows, Linux or ChromeBook computers, iOS or Android phones or other mobile devices like iPad. Use browsers like Safari, Chrome, Firefox or Edge. Since your App is mostly running in the Cloud, the user experience is surprisingly similar despite these diverse environments.

A Vendor should test an App across each supported environment to account for different screen sizes, supported technologies or browser interface differences. You might choose to deliver an App as separate computer or mobile editions to best accommodate screen, technology and user interface differences.

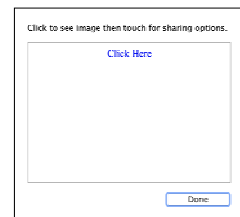
Screen size can be substantially different between computers, notepads or phones. To accommodate iPad users, a developer can set the Sheet Select Width parameter on the App Edit dialog so the top Sheet Selector is fully visible on the screen.

An ExcelRT app can use an HTML control that makes use of HTML, CSS, Javascript and other HTML technologies. Since those technologies run on the client side, capabilities can differ based on the browser or device where the App is running.

ExcelRT supports audio using a variety of formats. Support for audio formats may differ based on browser or client device.

One big difference between a computer or mobile device running an App is the user's ability to access files on that device. On a computer, the user can easily access files on disk for import or export into the App. Since a mobile device gives the user less access to files, file storage and sharing may depend more heavily on the Plugin folder in the user account or on a Cloud Sharing account used by the App.

Some ExcelRT script commands allow a browser based App to share files with Apps running on a mobile device. For example, the CloudViewer script command presents a dialog to upload images stored on an iPad or download and copy those images to the Photos or Files app.



The FileSaveAsFromPlugin script command can be used to download in image file from the Plugin folder. It presents a dialog with a Click Here link. On a computer, the image is immediately stored in the Downloads folder. On iPad, the downloaded image is displayed. The user can touch the image, then click **Save Image** or **Copy** in the presented Popup menu. Save the image to Photos or Paste it to the Files app.

ExcelRT Control Panel

An ExcelRT Cloud account is a process running on it's own processor core on an Internet server. That process services you the vendor when defining Apps and Users. It also presents each of your Apps for all User accounts. As each user logs into an App, memory is allocated to service that browser session.

ExcelRT provides a lot of power and freedom to the developer with a comprehensive scripting environment. With that power, comes responsibility. A developer can easily create a buggy script that crashes the application. When an ExcelRT Cloud process crashes, all user accounts are disabled.

The developer should fully test an App before presenting it to users. To maximize App availability, the ExcelRT Cloud process can be quickly restarted by the developer or with an automated process.

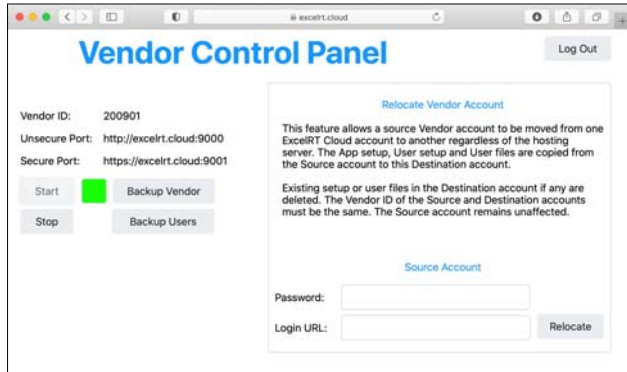
Each ExcelRT Cloud account comes with a Control Panel with its own Login screen.

Enter your Vendor ID in the User field and your account password then click **Login**.



The Control Panel screen shows a green box when your ExcelRT Cloud process is running or a red box when it is not running.

The developer can click the **Start** or **Stop** button to run or kill the process.



When clicking the **Start** or **Stop** button, it may take a few seconds for the box color to change. When the developer starts the process, the Control Panel process itself stays running even when you log out. It continually monitors your ExcelRT Cloud process. If it quits for any reason, it quickly restarts the process.

A developer may want to briefly stop their ExcelRT Cloud process for a scheduled upgrade or some other reason.

When a developer signs up for a new ExcelRT Cloud account, they get the current build of ExcelRT. Over time, new features will be added to ExcelRT but your running process and active users will be unaffected.

To use the new ExcelRT features, contact Excel Software to temporarily stop your process, replace your build of ExcelRT Cloud and restart the process.

When logged into an ExcelRT Cloud Vendor account, click the **Goto Control Panel** button to present the ExcelRT Control Panel login screen.

Secure and Unsecured Ports

An ExcelRT Cloud account is available from either a secure or unsecured port. Notice the link to each Login screen on the Vendor Control Panel.

Most users are more comfortable logging into a secured port so that may be the only Login URL that you will provide. With a secure port, communication is SSL secured and the user sees the familiar padlock icon in most browsers.

Your application runs exactly the same from an unsecured port. For large sheets, screen updates may be a bit faster. In the unlikely event that SSL security mechanisms of the Internet are temporarily offline, your Apps can still be accessed with an unsecured port.

ExcelRT Cloud Pricing

Each vendor must think through the business process and pricing for Apps running in a Desktop, Mobile or Cloud environment. Each environment has inherent differences that affect the development, distribution, support, license and price.

Development

The development process of an ExcelRT file is essentially the same regardless of the target platform. For smaller Apps, the same workbook developed and tested on a Mac or Windows computer can be distributed on any platform. Many scripting commands work across platforms, but there are some platform unique considerations.

Larger ExcelRT workbooks or those that require lots of data entry will perform better as a Desktop application due to the dedicated CPU and RAM of a local computer and potentially larger screen. Refer to the Design for Performance section of the Develop ExcelRT chapter.

Distribution

To distribute a Desktop app, a standalone Mac, Windows or Linux application can be created. The developer will likely create a Code Signed installer that installs the App and ExcelRT runtime on the customer computer. For Mac, the installer can be notarized with Apple for easily deployment on macOS Catalina. Refer to online ClickInstall information and videos for Mac and Windows desktop apps.

Desktop Apps are generally purchased from the developer website, downloaded to a local computer and then activated with an offline or Serial Number online activation process.

An iOS App is likely purchased and downloaded from Apple's App Store. For an Android App, that process can occur from an App Store or directly from the developer's website.

Cloud Apps are typically provided as part of a larger service or product bundle or purchased from the developer website. There is nothing to download and the user can run from a wide range of browsers on computer or mobile devices.

Support

The support cost for a Desktop app is generally higher than that of a Mobile or Cloud app since the runtime environment is more diverse.

Desktop users can have a diverse mix of Windows, Mac and Linux OS versions, CPU and RAM capabilities. The OS is more configurable and users have greater access to the software installed on their computer. That freedom gives the user the ability to mess something up.

Many users install Virus scanners or enable firewalls on desktop computers that affect their ability to download and run software. As a software vendor, you will likely incur higher support costs to deal with these types of user issues and questions.

Mobile Apps run in a more restricted, sandboxed environment. While initial development cost is likely higher than Desktop apps, ongoing support costs may be lower.

Cloud apps run in a browser. Most modern devices and computers should run your App fine since almost all processing and storage requirements occur on the server. ExcelRT Cloud or your App master workbook can be updated once on the server and is instantly available to all users. Support costs for Cloud apps should be lower than Desktop or Mobile apps.

License & Price

Desktop Apps are generally created with QuickLicense. For online Serial Number activation, a Safe Activation account can be used. Safe Activation can also support automated order processing, Serial Number delivery and license management.

Trial, Product and Subscription licenses are frequently used for desktop and mobile Apps. The same product can be offered as a lifetime purchase or monthly subscription. The same Serial Number in Safe Activation can be used for Desktop, Mobile and Cloud apps.

For a desktop app, the developer generally keeps 100% of the revenue. For a mobile distributed through an App store, the App Store by keep 30%. All iOS Apps are distributed through Apple's App Store so in addition to keeping a significant amount of the developer revenue, Apple imposes restrictions on the App developer.

With a Cloud app, the burden of providing CPU and RAM shifts from the local computer to the server. Most of the processing and memory requirements for a Cloud app occur on the server. A significant amount of RAM is allocated to each concurrent user session.

With a Cloud app, the Vendor is essentially providing a virtual computer for each active user session. For example, when a user session loads a compressed 1 MB workbook, 40 MBs of server RAM may be needed to support that session. RAM requirements grow quickly with dozens or hundreds of concurrent users.

Since a Vendor must directly or indirectly pay for the server CPU and RAM cost, the cost per user is higher for a Cloud app than for a Desktop or Mobile app. On the flip side, the user gets similar App performance regardless of their local computer or mobile device.

Since a vendor incurs an ongoing cost for each user account in the cloud, a time-limited or Subscription license is more applicable than a lifetime license so the recurring revenue matches the vendor's ongoing cost. ExcelRT Cloud can be used independently or linked to a Safe Activation account to manage licenses.

An ExcelRT Cloud account is priced as a scaleable monthly fee with no setup or cancellation fees. The monthly fee is based on the number of Apps, Users and Max file size allowed for each App workbook. The monthly cost largely reflects the cost of server RAM and CPU required to support your App users.

Goals of an ExcelRT Cloud account:

- Limited upfront technical knowledge for setup
- Low startup cost or financial risk for new apps
- Short concept to delivery time for new apps
- Scaleable cost paid from growing customer revenue
- Low monthly cost for small Apps with many users
- Low monthly cost for large Apps with few users

Multiple Vendor Accounts

Each ExcelRT Cloud account has a Vendor ID. Every developer gets a different Vendor ID. There are some situations where a developer may want multiple cloud accounts for the purpose of improving user performance, lowering cost or ensuring uptime of their Apps.

If a developer is building a simple App and then making it available to customers with few changes thereafter, they will likely only need one ExcelRT Cloud account.

Assume for example, the developer has created an App and now has many active users paying for a monthly Subscription. Perhaps the developer wants to create a new App or make major enhancements to the existing App. It is safer to develop and test new features, especially scripting commands using a separate Cloud account.

Assume the production build of your released App is running in a 1 MB account with thousands of user accounts. Assume you are developing a new version of that App that requires a 2 or 3 MB account. To develop and test the new App, it would cost less to get a separate account that supports larger files, but just a few users.

A developer can have multiple Cloud accounts using the same Vendor ID. Each Cloud account has a different Port number in the login URL as illustrated here.

`https://excelrt.cloud:9001`
`https://excelrt.cloud:9005`

If a developer has multiple accounts with the same Vendor ID, you'll need to differentiate the specific account when logging into the Control Panel by including the Vendor ID, a dash and the Port number as illustrated here:

`200115-9003`

Relocate Vendor Account

ExcelRT Cloud was designed to allow an entire Vendor account to be easily moved from one Cloud account to another even if different servers are involved. Refer to the Relocate Vendor Account section of the Vendor Control Panel screen. The original and relocated accounts must use the same Vendor ID.

This feature paves the way for future flexibility. Perhaps in the future you want to move your account to a different hosted server or even a self-hosted server. That can be accomplished with a few clicks. In minutes, all Apps and User accounts are running on a different server with a different URL.

Safe Activation

ExcelRT Cloud can be integrated with Safe Activation to manage user accounts. When an order occurs, a Transaction can be created in Safe Activation that assigns a Serial Number and delivers it to the customer in an email message. The customer data can also be communicated to ExcelRT Cloud to create a user account.

If the Vendor or an automated process later suspends the Serial Number or clears the Subscriber checkbox, that data can also be sent to ExcelRT Cloud. The associated App can be unassigned on the user account or an Alert can be set so the Vendor can take appropriate action.

A setup process is required in the Vendor account on both Safe Activation and ExcelRT Cloud. Communicated data is retained if ExcelRT Cloud is offline when the transaction occurs.

Create User Account

The licensing process within Safe Activation for Desktop, Mobile or Cloud app is driven by Serial Numbers. During the purchase process, the customer is provided with a Serial Number that can be used to activate a Desktop or Mobile app. The same Serial Number can be used to control multiple Apps running on multiple computers, devices or even in ExcelRT Cloud.

In an automated purchase process, a Transaction is created that assigns the Serial Number to a customer. That process can send the order data to ExcelRT Cloud so a user account can be automatically generated.

Here is the sequence of events:

1. Customer makes purchase (often with Paypal Subscription button).
2. Transaction is created on Safe Activation and email sent to customer.
3. Order data is sent to ExcelRT Cloud.
4. ExcelRT Clouds polls the received data and creates new User account.

On Safe Activation, the Order Edit page contains setup data to generate the transaction, assign and deliver the Serial Number and forward data to ExcelRT Cloud. To link your Safe Activation and ExcelRT Cloud accounts, follow the online help topic by including this line in the Notes field of the Order Edit page.

ExcelRTCloud:URL,CloudVendorID,OrderPassword

The URL is the URL used to log into your ExcelRT Cloud account. The CloudVendorID field is the Vendor ID assigned to your account. The OrderPassword field is shown on the Order Data dialog in your Vendor Account.

From the Vendor account in ExcelRT Cloud, click the **Order Setup** button to configure the account creation process.

Poll once a minute for order data sent from Safe Activation to generate new user accounts or assign new Apps to an existing account. If Vendor or User cancels the Subscription or Suspends the Serial Number, set an Alert on User account and/or Unassign the App. Set criteria for default User account name and password.

Order polling occurs after logging out of the Vendor account and quitting browser.

Date	App	First Name	Last Name	Order ID	Action	Serial Number
200209	BridgeScoreRank	John	Doe	123454321	NewUser Alert	01259-10012-78157-10101
200209	BridgeScoreRank	Jane	Doe	123454321	NewUser Alert	01259-14412-78177-10345
200209	BridgeScoreRank	Tom	Smith	123454321	NewUser Alert	01449-14466-78157-10645

☒ Poll for Orders
 ☒ Alert on Suspend
 ☒ Unassign on Suspend
 ☒ Alert on New User

Default Account Name:

☐ FirstnameLastname
 ☐ Serial Number
 ☐ Order ID
 ☐ FirstnameLast3OrderID

Default Password:

☐ Account Name
 ☐ Serial Number
 ☐ Order ID
 ☒ Firstname

App Family
 AppX=AppX1,AppX2,AppX3

Add
 Edit
 Delete

☐ Confirm
 Order Password: 14316

Set the Poll for Orders checkbox. To flag newly created user account for further action by the Vendor, set the Alert on New User checkbox.

Under Default Account Name, select a radio button to set the criteria used to construct the Account name. The account name can be the user's First and Last name such as JohnDoe. Some vendors upload specialized Serial Numbers into their Safe Activation account and use that as an Account name.

The Order ID is assigned during the order process. For a Paypal subscription, the Order ID is assigned by Paypal and delivered to the customer in an email message.

The FirstnameLast3OrderID option uses a combination of the customer's first name plus the last 3 digits of the assigned OrderID making it very likely to be unique.

When a transaction occurs, an email is sent from Safe Activation to the customer. It includes the OrderID, Serial Number and can instruct the user on how to log into their new ExcelRT cloud App by providing the App name, User name and default Password.

The Default Password is constructed based on the selected criteria. After logging in, the user can change the password as desired. The Vendor can also look up or change the password.

Suspend User Account

The Vendor or an automated process can suspend or a Serial Number or cancel a Subscription. An event can be configured in Safe Activation to set an Alert in a User account or unassign an App when the Suspend checkbox is set or the Subscriber checkbox is cleared for a Serial Number.

To configure an Event in Safe Activation, click the **Add** button in the Event menu to present the Event Edit screen.

The screenshot shows the 'Vendor Account' interface in a web browser. The main heading is 'Vendor Account' in large blue letters. Below it, the 'Event Edit' screen is displayed. The left sidebar contains a navigation menu with categories: General, Products, Custom Forms, Serial Numbers, Customers, Order Setup, Events, Downloads, Web Apps, and License Block. The 'Event Edit' form includes the following fields and options:

- Event Name:** Suspend User
- Enable Automated Processing:** ☒
- Loop Through:** ☒ Serial Number ☐ Customers ☐ Orders
- Rules:**
 - Rule A: Serial Number (dropdown), In Group (dropdown), BridgeScoreRankSN (dropdown)
 - Rule B: Subscriber (dropdown), Equals (dropdown), Nn (dropdown)
 - Rule C: Mark (dropdown), Equals (dropdown), No (dropdown)
 - Rule D: Undefined (dropdown), Undefined (dropdown), Undefined (dropdown)
 - Rule E: Undefined (dropdown), Undefined (dropdown), Undefined (dropdown)
- Condition:** A and B and C (with a note: 'may contain A,B,C,D,E,and,or,space character')
- Action:** Suspend ExcelRT Cloud User (dropdown)
- Email Actions:**
 - ☐ UTF8 Encoding ☐ HTML Formatting ☐ Include Names
 - From Name:** (text field) **To Name:** (text field)
 - From Email:** excelrt@spinn.net
 - Remote Emailer:** (text field)
- Email and Notification Actions:**
 - Subject:** (text field)
 - URL:** https://excelrt.cloud/8000s/191219.BridgeScoreRank/56/99

Refer to the help topic in Safe Activation to configure an event that triggers data to be sent to ExcelRT Cloud when the Suspend checkbox is set or the Subscriber checkbox is cleared.

Cloud Apps are generally sold as monthly subscriptions, where the batch of Serial Numbers has the Subscriber checkbox initially set. As long as the periodic subscription is paid, the user remains an active subscriber.

From the Order Data dialog in ExcelRT Cloud, set the Alert on Suspend and/or Unassign on Suspend checkbox to take appropriate action if the account is suspended or the subscription is canceled.

App Family

One Serial Number can control a family of Apps when creating the user account, assigning new Apps to an existing user account or suspending Apps.

On the Order Data screen, click the **Add** button next to the App Family listbox.

In the App Family dialog, enter the name for a family of Apps. Now enter a comma-separated list of App names.

Within Safe Activation, the purchased product is the App Family. When a user purchases a Serial Number an account can be created in ExcelRT Cloud with all of the assigned Apps. Later those Apps can be suspended using that same Serial Number.

A family of Apps can be controlled by one Serial Number. When purchased, a user account is created with all assigned Apps. When suspended, all Apps can be unassigned or an Alert set.

App Family Name:

Comma Separated App Names:

ExcelRT Plugin

An ExcelRT Plugin file can be used in an ExcelRT Cloud account in several ways. To an application user, each approach appears to work the same. For a developer, there can be a significant impact during development, testing and for ongoing maintenance.

To understand each approach, consider that each user account has a separate Plugins folder. All Apps within an account share the same Plugins folder. If 500 user accounts are assigned to your App and the Plugin file is stored in each Plugins folder and is permanently expanded into each Plugins, a lot of disk space is required and maintenance gets complicated if you want to change files within the Plugin.

A Shared plugin with on-demand virtual instances can offer huge advantages.

Manual Expand

The Plugin file is stored in the Plugins folder for each user account. Using a script command, the Plugin file can be expanded into a folder. The HTML control and PluginTaggedTemplate script command references the expanded folder name.

Auto Expand

The Plugin file is stored in the Plugins folder for each user account. The HTML control and PluginTaggedTemplate script command references the Plugin file name rather than the expanded folder name. On first use, if the Plugin file is not password protected, it is expanded to a folder within the Plugins folder.

Virtual Expand

The Plugin file is stored in the Plugins folder for each user account. The HTML control and PluginTaggedTemplate script command references the Plugin name rather than the expanded folder name. On first use, if the Plugin file is password protected, it is expanded to a virtual folder. The virtual folder is deleted when the browser session ends.

Shared

The Plugin file is stored once in the vendor account and never copied into the Plugins folder of any user account. The HTML control and PluginTaggedTemplate script command references the Plugin file name rather than the expanded folder name. When an App runs in a user account that references the Plugin, a virtual instance of the expanded plugin folder is created and retained until the browser session is closed.

As a developer, you are probably wondering how and when a Plugin file is copied into the Plugins folder of each user account or how to designate a Shared plugin.

From the App Edit dialog, a Plugin file can be uploaded just like the ERT or XML file itself. With that App selected on the Vendor page, click the **Plugins** button to present a dialog that designates how files are managed within user accounts.

The screenshot shows a dialog box titled 'Plugins' with a table of plugin files. The table has two columns: 'Filename' and 'Copy to New User Plugin Folder'. The files listed are BarChart1.excelrt_plugin, BarChart2.excelrt_plugin, BarChart3.excelrt_plugin, HTML1.excelrt_plugin, HTML2.excelrt_plugin, Landscapes.excelrt_plugin, and LineChart.excelrt_plugin. The 'Copy to New User Plugin Folder' column shows 'Shared' for BarChart1, BarChart2, BarChart3, and Landscapes; and 'Yes' for HTML1 and HTML2. BarChart2.excelrt_plugin is highlighted in blue. Below the table are several buttons: 'Copy To User ID', 'Renamed To:', 'Delete', 'Copy To All Users', 'Copy Yes', 'Copy No', 'Shared', 'Replace Master for Assigned Users', and 'Done'.

Filename	Copy to New User Plugin Folder
BarChart1.excelrt_plugin	Shared
BarChart2.excelrt_plugin	Shared
BarChart3.excelrt_plugin	Shared
HTML1.excelrt_plugin	Yes
HTML2.excelrt_plugin	Yes
Landscapes.excelrt_plugin	Shared
LineChart.excelrt_plugin	Shared

Copy To User ID Renamed To:

Delete Copy To All Users Copy Yes Copy No Shared

Replace Master for Assigned Users Done

In this example, BarChart2.excelrt_plugin is marked as Shared and HTML1.excelrt_plugin is marked as Yes indicating that this file is copied into the Plugins folder of each newly created user account. After setting a file as Shared (or changing to not Shared), go to the ExcelRT Control Panel and Stop and Start the server for the change to take affect.

When an App is assigned to a user account on the Vendor page, files marked as Yes in the dialog above are copied to the Plugins folder of that account. Files marked as Shared are not copied.

For existing user accounts that are already assigned to an App, marking a new file as Yes will not cause the file to be copied to the Plugins folder. The **Copy To User ID** or **Copy To All Users** buttons could accomplish that.

A Vendor can see what files are currently stored in the Plugins folder of a selected user account by clicking the **Plugins** button above the user list. From this dialog, the Vendor can delete a file or folder of files from the user account Plugins folder.

For special situations, a Plugin file can be copied into the Plugins folder of a user account using a Cloud Sharing account or even retrieved from a Vendor website using script commands.

User Profile

The **Profile** button presents the User Profile dialog for a selected user account. This dialog shows which Apps a user runs, how often and the screen size used.

Cookie	Width	Height	Apps
20210722_19_49_25	1370	996	QuoteX:26,Test:8,Test2:7
20210728_13_24_48	1456	1165	Test:1
20210728_14_50_04	954	768	Test:2,Quote:1,CP:2
20210728_15_07_38	854	600	Test:2,Test2:5,Quote:3,ChartTest:1,CP:3
20210729_17_47_05	600	400	Test2:1
20210802_14_35_48	1213	1007	Test:1,Quote:1
20210802_14_37_37	1080	556	Test:2,Quote:1
20210814_13_54_59	1533	836	ChartTest:1,Plugin:2

Clear

Done

When an ExcelRT app runs from a browser for the first time, it creates a Cookie that stores the current date, hour, minute and second. Other information collected during the session includes the screen size. Together these fields form a profile that is relatively static over the life of an App.

In the example above, each row represents a specific device and browser combination. If an App runs multiple times on that browser, that row of data can update over time. For example, the Width, Height and Apps fields of a specific row represent a composite of information collected over time.

Most computer screens are wider than they are tall. On a computer, the user can resize the browser window width and height. Mobile devices can often be rotated but the browser width and height is fixed. For each user session, the longest axis is assumed to be the width and the shortest axis is assumed to be the height. On a future session, if the width or height is increased, the larger value is stored on that profile line.

For a mobile device, the Width and Height instantly shows the vendor the size of the browser window. For an App running in a computer browser, over time the Width and Height parameters show the maximum browser size used to present the App.

The Apps field shows the name of each App that runs in that user account. Notice each App name is followed by a : and the launch count for that App. When multiple Apps run from the same user account, App names are separated with commas.

Script Commands

ExcelRT supports hundreds of script commands. Almost all commands work exactly the same for either macOS or Windows desktop applications built with ExcelRT. There are a few platform specific commands that only apply to macOS or Windows.

Most ExcelRT commands work the same in ExcelRT Cloud as they do in a desktop application. The environment of a cloud application is inherently different since the user doesn't have direct access to the hard drive where the application is running.

Script commands that apply to a specific OS are not supported in ExcelRT Cloud. Some commands that integrate with a specific technology like ODBC are not supported. Another such example is Python integration. While the Python command is not supported, the PythonServer command is supported with the ExcelRT Vendor Commands suite of script commands.

Script commands that are not supported in ExcelRT Cloud can either be ignored or an error dialog can be presented based on the Ignore Unsupported Script Commands checkbox on the App Edit dialog used to define the application. When testing a script in ExcelRT Cloud that was originally developed for a desktop application, the developer may want to see a message for each supported command. After making necessary script changes, this checkbox can be set to ignore those unsupported commands.

HTML controls and Plugins may require some changes in ExcelRT Cloud. If the HTML content is coming from the Internet, it should work essentially the same in a desktop or cloud app. Assume the HTML content is coming from the Plugins folder. In a desktop application, the main HTML file can reference other files at a relative location. In a cloud application, there is a Plugins folder that can hold the main HTML file, but references to other files at a relative location are broken. For a complex HTML control, put the files on your own website and then reference the main URL from the HTML control on an ExcelRT sheet.

Backups

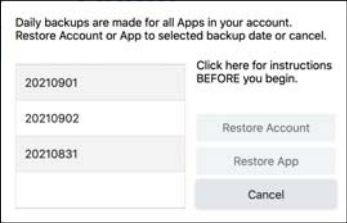
ExcelRT Cloud implements a multi-tier backup strategy. Each day, a backup is made of the entire Vendor account and individual backups are made of each user account. These rolling backups are stored for three days.

Here are the backup types:

- **Vendor Backup** – A Vendor can restore their entire account including all user-generated files to the state it existed during one of the last three backups.
- **User Backup** – A User can restore the files associated with a specific App or with their entire account (an account may have several assigned Apps) to a backup made during the last three days.
- **Cloud Sharing Account** – Users can upload and download files to an external Cloud Sharing account. A Cloud Sharing account allows files to be shared with other users, desktop applications or archived on a local Mac or Windows computer.
- **Server Backup** – Excel Software maintains a daily archive of Vendor accounts. While technically possible, the manual restoration process will cost the vendor hundred dollars to cover the human time involved.

Here is the dialog presented by clicking the **Restore** button in a user account.

Users have the ability to quickly restore their account to a previous backup if they accidentally delete an important file.



Daily backups are made for all Apps in your account. Restore Account or App to selected backup date or cancel.

20210901	Click here for instructions BEFORE you begin.
20210902	
20210831	

Restore Account

Restore App

Cancel

Chapter

9

Tutorial

To create a Mac or Windows application from your Excel workbook file requires multiple steps and several tools. It's a process that may take several days depending on the complexity of your workbook.

Before starting on your own project, you need a basic understanding of how the whole process works. The TravelCalc tutorial starts with an Excel workbook, converts it to an ExcelRT file and then adds licensing and user interface options to produce a protected standalone application.

TravelCalc

Right-click on the ConvertExcelRT shortcut icon and choose **Show File Location** from the popup menu. Locate the Sample1 folder. If your Windows OS is not displaying file extensions, use View Options to turn file extensions on.

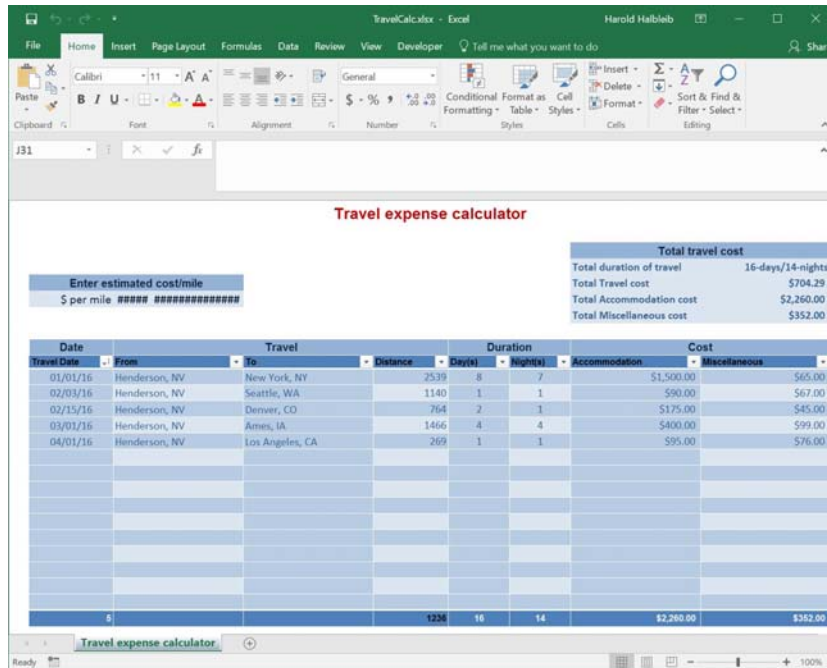
Open Workbook in Excel

Open `TravelCalc.xlsx` into Microsoft Excel. You will see a simple workbook with one sheet that allows rows of information to be entered for travel expenses. A table of alternating background colors holds much of the data. Some basic calculations are performed on the entered data.

Notice how the column and row headers have been hidden. The workbook developer has applied a cell format to each cell. Once converted, data gets presented nicely within ExcelRT.

Notice how data is centered in some columns and right justified in others. Unlike Excel, ExcelRT doesn't try to guess how you want data formatted or justified.

Notice the title “Travel expense calculator” in red across the top. For that text to span several cells in ExcelRT, select those cells and choose Merged Cells in the Format dialog.



Workbook in Microsoft Excel

Generate TravelCalc.xml

In this section, you will convert the Excel workbook into an XML file that can be opened into ExcelRT using the ConvertExcelRT tool.

Your user account probably does not have read/write access to files stored under the Program Files directory structure. Copy file `TravelCalc.xlsx` to a Test folder within your Documents folder.

Launch ConvertExcelRT and set options in the main window so they match those shown below. Ignore options related to pictures since this workbook has no pictures. Make sure the Overview checkbox is set, then close the window to quit the application and save your changes.

Open TravelCalc in Design Mode

To open TravelCalc.xml into ExcelRT, drag and drop it on the application icon or shortcut file. The workbook is presented within the ExcelRT application.

Travel expense calculator

Enter estimated cost/mile
5 per mile \$0.57

Total travel cost	
Total duration of travel	16-days/14-nights
Total Travel cost	\$3,521.46
Total Accommodation cost	\$2,260.00
Total Miscellaneous cost	\$352.00

Date	Travel		Duration			Cost	
Travel Date	From	To	Distance	Days	Nights	Accommodation	Miscellaneous
01/01/16	Henderson, NV	New York, NY	2539	8	7	\$1,500.00	\$65.00
02/03/16	Henderson, NV	Seattle, WA	1140	1	1	\$90.00	\$67.00
02/15/16	Henderson, NV	Denver, CO	764	2	1	\$175.00	\$45.00
03/01/16	Henderson, NV	Ames, IA	1466	4	4	\$400.00	\$99.00
04/01/16	Henderson, NV	Los Angeles, CA	269	1	1	\$95.00	\$76.00

Workbook in Design Mode within ExcelRT

Type a new row of data into the workbook. Notice how calculations are updated. Sheet names are listed across the top of the window. For this workbook, the column and row labels and scrollbars have been hidden.

At the top-left corner of the window, notice the small ribbon of tools. This can also be hidden if it presents no value to the application. Since an XML file is open, ExcelRT is in design mode.

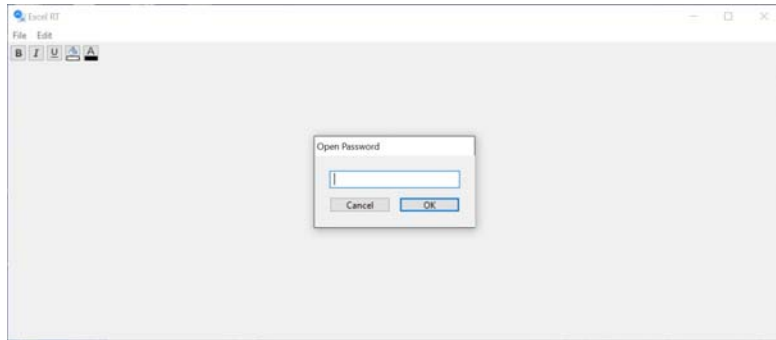
In Design mode, the **File** menu has several commands. Most of these commands are not used by a typical developer, but can be useful when troubleshooting a conversion issue.

One command you will use is the **Save Encrypted** command. It outputs an encrypted ERT file suitable for a customer. For this tutorial, we have created the ERT file for you. Click the close box in the top corner of the ExcelRT window to quit the application.

Open TravelCalc in User Mode

The Sample1 folder supplied with ConvertExcelRT includes TravelCalc.ert with an Open password of “Test” applied to it. Copy that file to your Test folder, then drag and drop that file onto ExcelRT. You can also launch ExcelRT and it will prompt you for the file to open.

Notice the Open Password dialog is presented because that feature was applied to the encrypted file. Enter “Test” without the quotes and click **OK**.



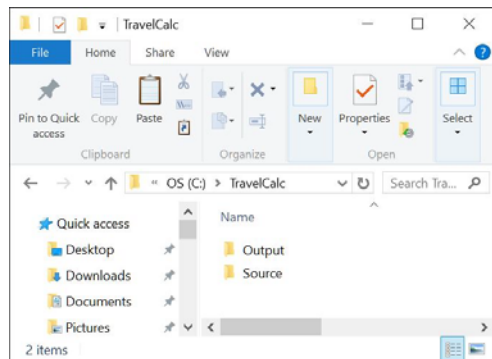
Open Encrypted File into ExcelRT

Once the file is open, everything looks about the same in User mode as it did in Design mode. Notice how we choose to hide the ribbon. Also, notice how most commands on the **File** menu have disappeared.

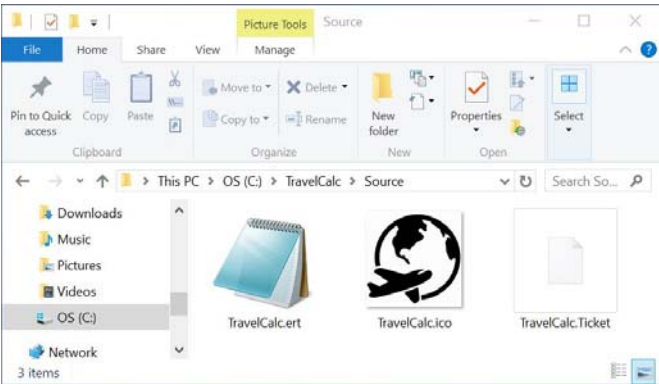
Add License to App

Most developers that sell an application using ExcelRT will want to apply a license for computer unique activation. QuickLicense supports many license types, activation procedures and a customizable interface for your app.

To complete this tutorial, you'll need QuickLicense. Even without it, you can follow along to see how the process works. Create a TravelCalc folder directly on drive C and inside create a Source and Output folder as illustrated here.



Copy several files to the Source folder that are used to generate the protected application. TravelCal.ert is the file used to generate the app. TravelCalc.ico is a windows icon that will be applied to the application. On Mac, use TravelCalc.icns.

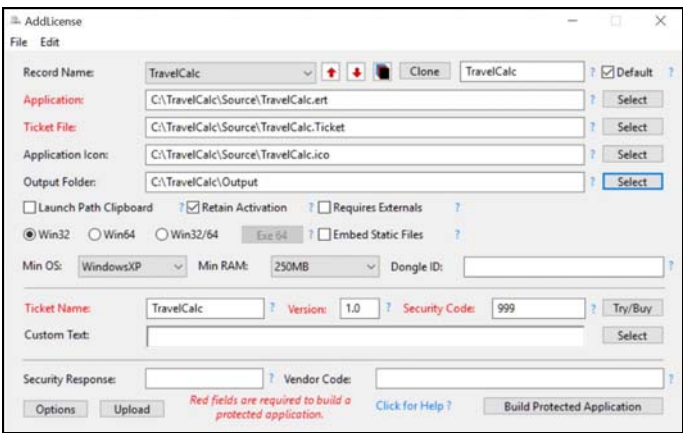


Source Files for Protected Application

Icon files can be created from a graphic image using an Icon building tool, but are included in the Sample1 folder for your convenience.

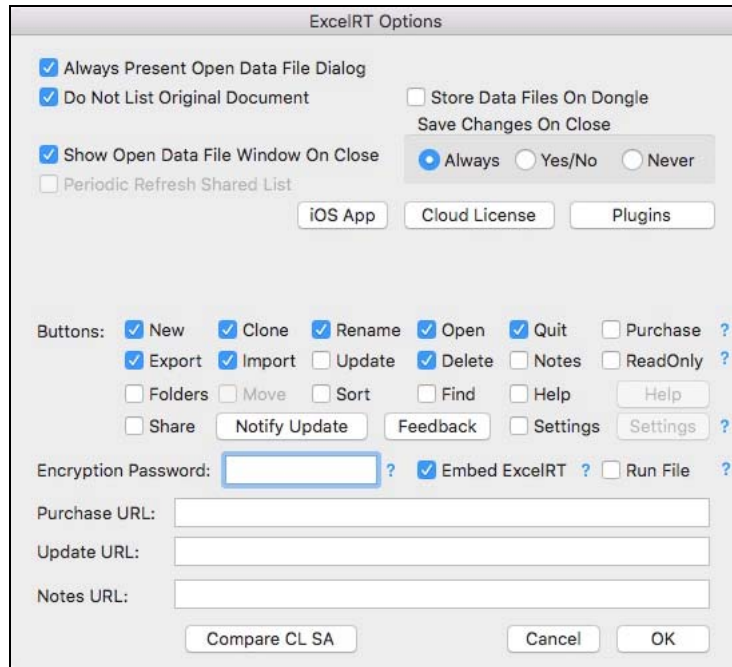
TravelCalc.Ticket defines a simple product license and pre-activation license agreement configured with QuickLicense. After completing this tutorial, you may want to review Tutorial 1 that comes with QuickLicense for details on how to produce your own custom Ticket file.

Launch the AddLicense wrapping tool include with QuickLicense.



Build Protected App with AddLicense

Within the main AddLicense window, select your source files and the output folder as illustrated above. Click the **Options** button and present the ExcelRT Options dialog.

The image shows the 'ExcelRT Options' dialog box. It has a title bar 'ExcelRT Options'. Inside, there are several sections. The first section has checkboxes: 'Always Present Open Data File Dialog' (checked), 'Do Not List Original Document' (checked), 'Show Open Data File Window On Close' (checked), and 'Periodic Refresh Shared List' (unchecked). To the right of these is a group box 'Save Changes On Close' with radio buttons 'Always' (selected), 'Yes/No', and 'Never'. Below this are three buttons: 'iOS App', 'Cloud License', and 'Plugins'. The second section is labeled 'Buttons:' and contains a grid of checkboxes for various actions: 'New' (checked), 'Clone' (checked), 'Rename' (checked), 'Open' (checked), 'Quit' (checked), 'Purchase' (unchecked), 'Export' (checked), 'Import' (checked), 'Update' (unchecked), 'Delete' (checked), 'Notes' (unchecked), 'ReadOnly' (unchecked), 'Folders' (unchecked), 'Move' (unchecked), 'Sort' (unchecked), 'Find' (unchecked), 'Help' (unchecked), 'Share' (unchecked), 'Notify Update' (button), 'Feedback' (button), 'Settings' (button), and 'Settings' (button with a question mark). The third section is 'Encryption Password:' followed by a text input field (highlighted with a blue border) and a question mark. To the right are checkboxes 'Embed ExcelRT' (checked) and 'Run File' (unchecked), both with question marks. The bottom section has three text input fields: 'Purchase URL:', 'Update URL:', and 'Notes URL:'. At the very bottom are three buttons: 'Compare CL SA', 'Cancel', and 'OK'.

Configure App Interface with ExcelRT Options Dialog

Most workbooks are designed to allow a user to create different named copies of the file for each client or project. For the TravelCalc app, the data for each employee is stored in a named copy of the data file as you'll see later in the tutorial.

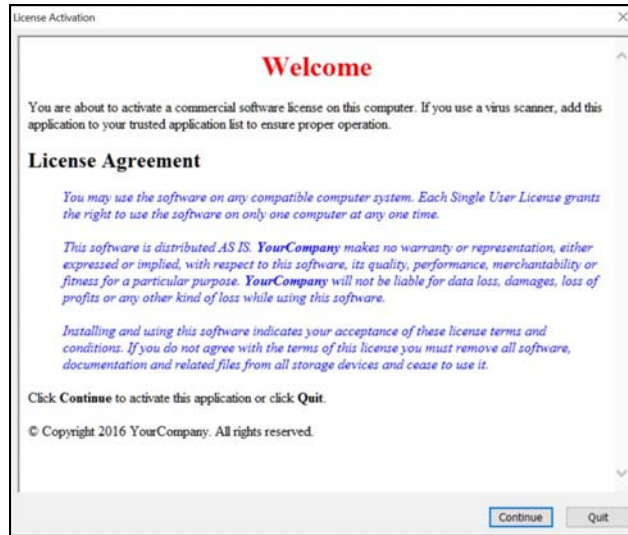
Notice the checkboxes named New, Clone, Rename, etc. These are used to customize the user interface of the Open Data File interface window presented by the protected application. If you want your application to always open the same data file, you can completely suppress the Open Data File window.

Fill in the Encryption Password field to match the Open password that was applied to your encrypted file, then click **OK** to close the dialog.

Click the **Build Protected Application** button in the main AddLicense window to generate the protected application with a custom icon.

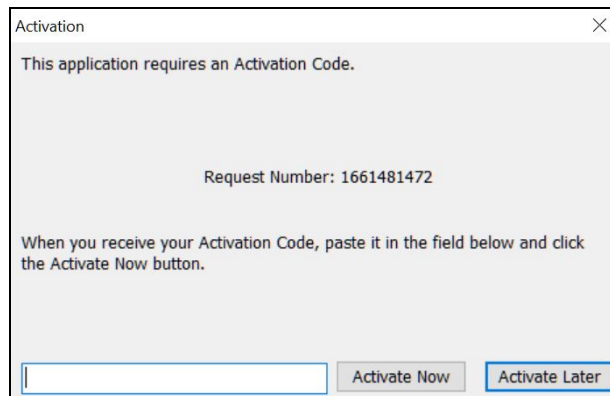
Launch Protected App

The protected application generated in the Output folder can be renamed and distributed to each customer. On launch, a license agreement is presented. This is one of the many optional features in QuickLicense.



License Agreement Presented Before Activation

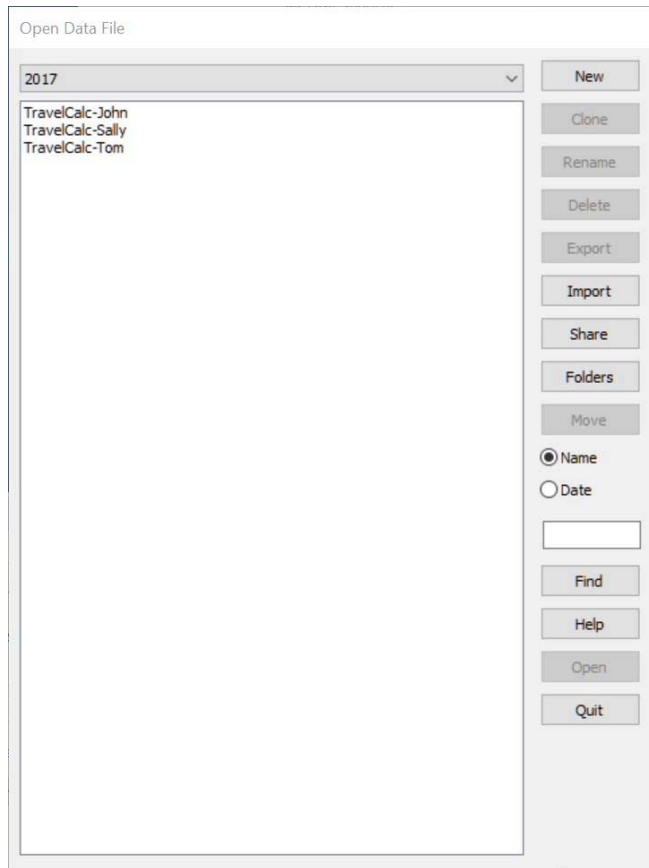
When the user clicks **Continue**, a manual Activation dialog is presented. You might choose to use an online Serial Number activation process or USB dongle instead.



Manual Activation Dialog

The Request Number displayed in the Activation dialog is unique for each computer. For the purpose of this tutorial, we've made the Activation Code the same as the displayed Request Number. Type that number and click **Activate Now**.

The Open Data File window is presented. Keep in mind, you can customize this window by hiding any buttons you don't need in your app. This window is presented each time the user launches your app to provide a user interface for managing data files.



Open Data File User Interface Window

Click **New** to create a named file in the list. To open that file into ExcelRT, select it and click **Open** or just double-click.

Convert Workbook

This tutorial covers some of the most common issues faced by every developer when converting their workbook to run within ExcelRT. Complete this tutorial to demonstrate errors and solutions before attempting to convert your own workbook.

Create Simple Workbook

Create a new Excel workbook named Tutorial.xlsx then enter this data into the specified cells. When completed, your workbook should look like the image below.

1. In A1, type: Calculate Bank Account Interest
2. In A3, type: Account
3. In B3, type: Principle
4. In C3, type: % Rate
5. In D3, type: Interest
6. In D4, type: $=B4*C4$
7. In D5, type: $=B5*C5$
8. In D6, type: $=B6*C6$
9. In D7, type: $=B7*C7$
10. In D8, type: $=B8*C8$
11. In B10, type: Total Interest
12. In D10, type: $=SUM(D4:D8)$

	A	B	C	D
1	Calculate Bank Account Interest			
2				
3	Account	Principle	% Rate	Interest
4				0
5				0
6				0
7				0
8				0
9				
10		Total Interest		0
11				

Simple Excel Workbook

Convert Workbook

Launch ConvertExcelRT. Set every checkbox except for Overview and Maximize, then click the close box at the top right corner of the window to quit the application.

Drag and drop, Tutorial.xlsx onto the ConvertExcelRT application or shortcut icon. The conversion process should take a few seconds and outputs file Tutorial.xml. Drag and drop Tutorial.xml onto the ExcelRT application.



	A	B	C	D
1	Calculate. nterest			
2				
3	Account	Principle	% Rate	Interest
4				0.00
5				0.00
6				0.00
7				0.00
8				0.00
9				
10		Total .rest		0.00

Workbook in ExcelRT – Conversion 1

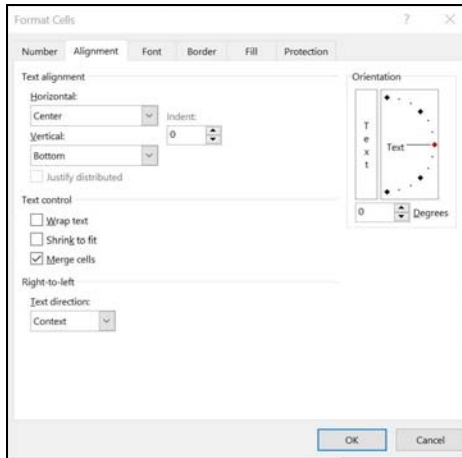
At first glance, you'll notice several issues. None of the cells are editable. When you click on a cell, the cell is not selected and you cannot type data into it. Let's fix that.

By default, Excel locks every cell but the lock doesn't take affect unless you protect the sheet. This allows the designer to control which cells have editable data and which cells are static labels.

The Lock feature will be useful later in the conversion process, but for now disable that feature from ConvertExcelRT. Launch ConvertExcelRT. Uncheck the Use Cell Lock checkbox then close the window.

The text labels in A1 and B10 in ExcelRT do not extend across cell boundaries. ExcelRT never extends text outside of a specific cell unless it's a merged cell.

Open Tutorial.xlsx and select cells A1:D1. Right-click on cell A1 and select the **Format Cells** command.



Merge Cells with Format Cells Dialog

On the Alignment panel, set the Merge Cells checkbox and for Horizontal alignment, select Center. On the Font panel, choose Bold and 16-point size. Choose red for the font color and click **OK** to save your changes.

Merge cells B10 and C10. Save and close the Excel document, run the conversion again and open the new XML file into ExcelRT. Now the labels look nicer and data can be entered into cells.

	A	B	C	D
1	Calculate Bank Account Interest			
2				
3	Account	Principle	% Rate	Interest
4				0.00
5				0.00
6				0.00
7				0.00
8				0.00
9				
10		Total Interest		0.00

Workbook in ExcelRT – Conversion 1

Cell Formats

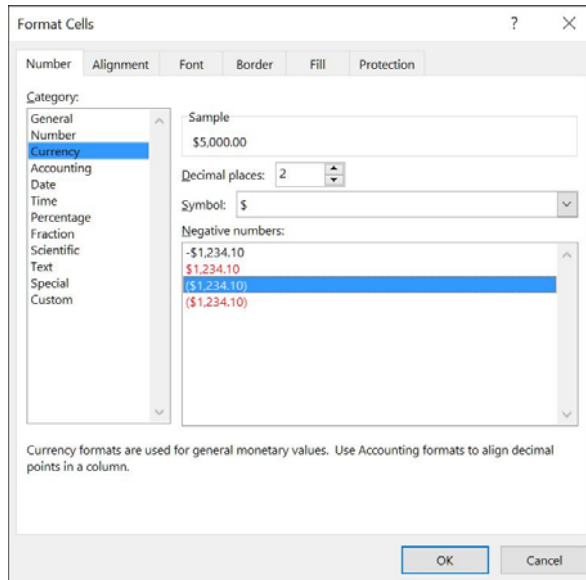
If you enter data into this workbook from Excel or ExcelRT, you'll find there is room for improvement with the addition of some cell formats and validation rules.

Open the workbook in Excel. Select cells A3:D3 and use the Format Cells dialog to set text to Bold, Underlined and Centered. Set the Font color to Blue.

Select cells B4:B8 and present the Cell Format dialog. Set the Currency format as illustrated below, then click **OK**.

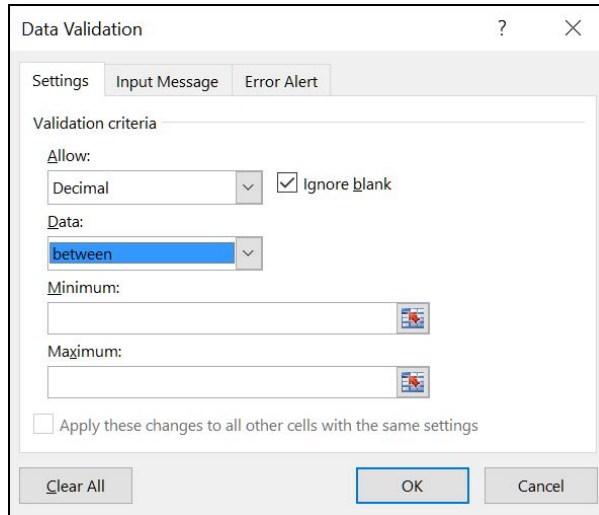
Select cells C4:C8 and present the Cell Format dialog. Set the Percent format and click **OK**.

Select cells D4:D8 and present the Cell Format dialog. Set the Currency format and click **OK**.



Select Currency Format

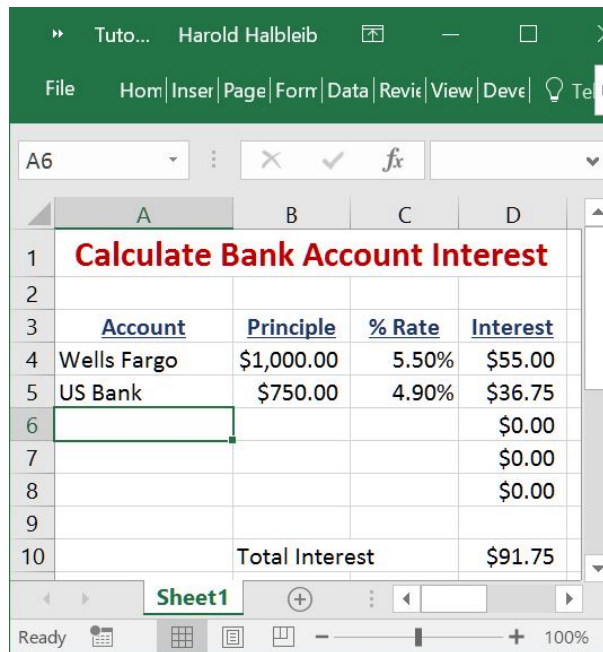
Select cells B4:B8 and present the Data Validation dialog. Restrict the user to entry of decimal values as illustrated below, then click **OK**. Select cells C4:C8 and restrict data entry to decimal values.



The image shows the 'Data Validation' dialog box in Microsoft Excel. The 'Settings' tab is selected. Under 'Validation criteria', the 'Allow' dropdown is set to 'Decimal', and the 'Ignore blank' checkbox is checked. The 'Data' dropdown is set to 'between'. The 'Minimum' and 'Maximum' fields are empty, each with a small icon to its right. At the bottom, there is an unchecked checkbox labeled 'Apply these changes to all other cells with the same settings'. The 'OK' button is highlighted with a blue border.

Use Data Validation To Restrict Data Entry

Add a few rows of data, then save and close the Excel workbook.

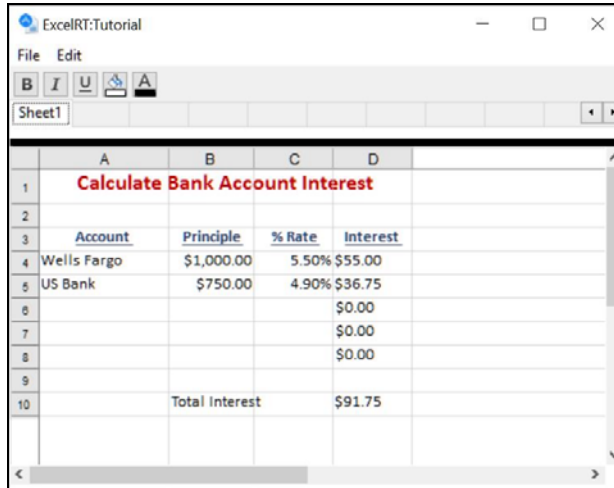


The image shows an Excel spreadsheet titled 'Tuto... Harold Halbleib'. The spreadsheet has a green header row and a table with 4 columns: Account, Principle, % Rate, and Interest. The table contains data for Wells Fargo and US Bank, and a total interest calculation. The 'Interest' column is formatted with currency symbols and two decimal places. The 'Total Interest' is calculated as \$91.75.

Account	Principle	% Rate	Interest
Wells Fargo	\$1,000.00	5.50%	\$55.00
US Bank	\$750.00	4.90%	\$36.75
			\$0.00
			\$0.00
			\$0.00
Total Interest			\$91.75

Excel Workbook with Data and Formatting

Convert and open the XML file into ExcelRT.



The screenshot shows a window titled 'ExcelRT:Tutorial'. The menu bar includes 'File' and 'Edit'. The toolbar contains icons for Bold (B), Italic (I), Underline (U), a link icon, and a text color icon (A). The sheet is named 'Sheet1'. The spreadsheet content is as follows:

	A	B	C	D
1	Calculate Bank Account Interest			
2				
3	<u>Account</u>	<u>Principle</u>	<u>% Rate</u>	<u>Interest</u>
4	Wells Fargo	\$1,000.00	5.50%	\$55.00
5	US Bank	\$750.00	4.90%	\$36.75
6				\$0.00
7				\$0.00
8				\$0.00
9				
10		Total Interest		\$91.75

Workbook with Formatting in ExcelRT

Presentation

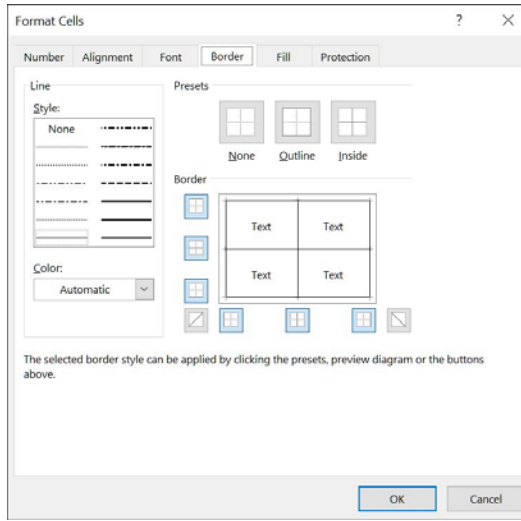
The workbook still has some presentation and usability issues. Notice how cells in the Interest column are left justified. All cells are editable including those with labels or those that are unused. For a finished application, only cells A4:C8 should be editable.

Open the `Tutorial.xlsx` in Excel and right justify cells D4:D10 from the Alignment panel of the Format Cells dialog.

For cosmetic appearance, resize row 1 so it about twice as tall as the other rows. Make columns A, B and D wider so they can display larger numbers.

To control which cells are editable and which are static, launch the ConvertExcelRT application and set the Use Cell Lock checkbox, then quit the application. Select cells A4:C8 and present the Format Cells dialog for that selected range. On the Protection panel, clear the Locked checkbox and click **OK**.

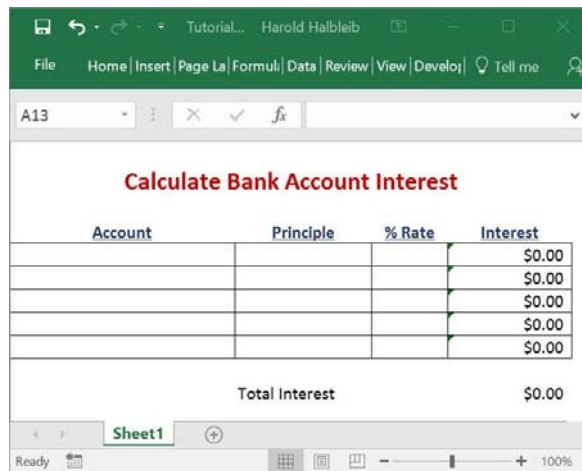
Select cells A4:D8 and present the Format Cells dialog. On the Borders panel add a thin line border around each cell and click **OK**.



Add Border to Editable Cells

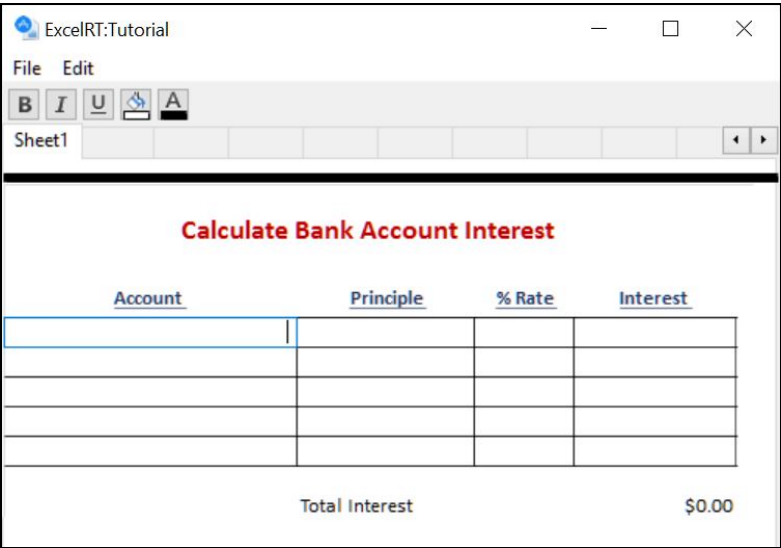
Choose the **Options** command from the **File** menu. On the Advanced panel, clear Show Horizontal Scroll Bar and Show Vertical Scroll Bar checkboxes. Also clear the Show Row and Column Headers and Show Grid Lines checkboxes.

Select and remove sample cell values you have entered without changing the cell format. Save the Excel workbook and compare it to the image below.



Finished Workbook in Excel

Convert the workbook and open it into ExcelRT. Enter data to verify calculations and ensure only desired cells are editable.



The screenshot shows a window titled "ExcelRT:Tutorial" with a menu bar containing "File" and "Edit". Below the menu bar is a toolbar with icons for Bold (B), Italic (I), Underline (U), a link icon, and a text color icon (A). The sheet is named "Sheet1". The spreadsheet content is as follows:

Calculate Bank Account Interest			
<u>Account</u>	<u>Principle</u>	<u>% Rate</u>	<u>Interest</u>
Total Interest			\$0.00

Finished Workbook in ExcelRT

Your Workbook

When you are ready to process your own workbook, start by writing down the title of each sheet and minimum required columns and rows of that sheet. For example, your list might be something like Sheet1-D7, Sheet2-N30, etc.

Launch ConvertExcelRT and set the Overview checkbox. Now process your workbook and when the Overview window appears, limit the cell range of each sheet based on your written list.

By minimizing the processed cells, the file size and processing time can be dramatically reduced.

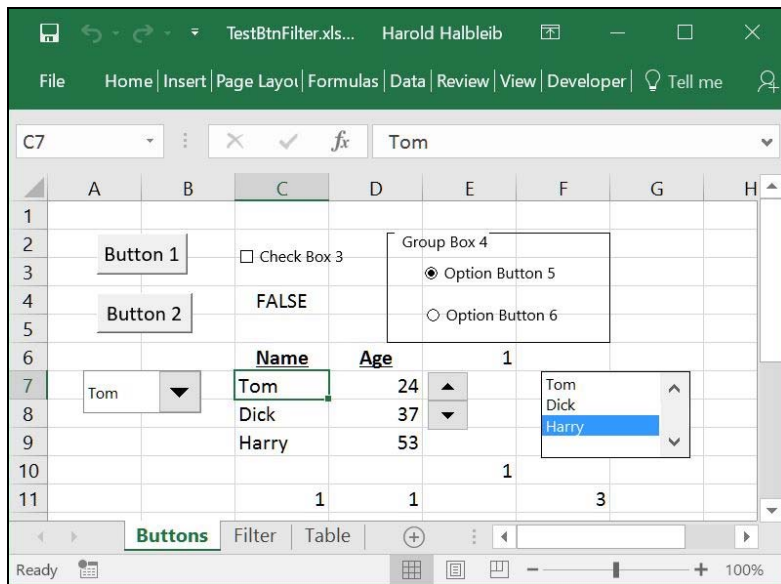
Controls and Scripts

This tutorial covers sheet filters and tables. It also uses form controls, scripts and programming to implement more advanced features that you may have previously used VBA code to accomplish.

The features described in this tutorial work essentially the same on Mac, Windows or Linux. Your screens may appear slightly different based on the platform you use.

Review Workbook

Open TestBtnFilter.xlsx into Microsoft Excel to review the source document used for this project. That file has been converted to TestBtnFilter.xml for this tutorial. Notice the workbook consists of three sheets named Buttons, Filter and Table.



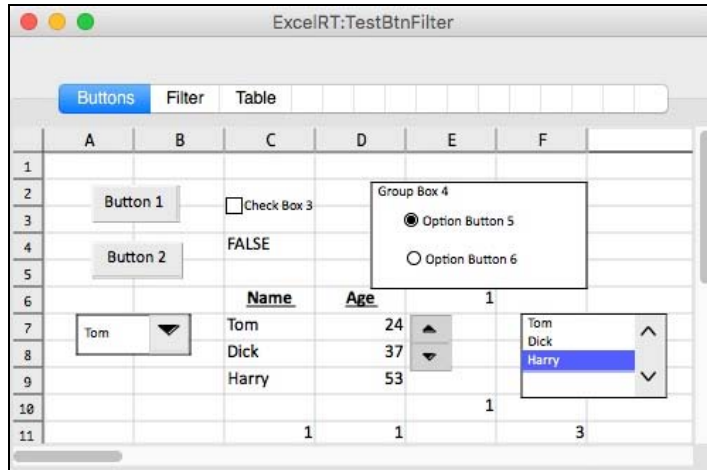
TestBtnFiler.xlsx from Sample4 Folder

While running this tutorial, you may want to leave this Excel workbook open so you can review how each control was configured within Excel and how it works within ExcelRT.

Open File in ExcelRT

File TestBtnFilter.xml was generated from the Excel workbook. It contains several types of Form Controls that were created from Excel's Developer ribbon.

Open TestBtnFilter.xml into ExcelRT by dragging and dropping it on the ExcelRT application icon.



TestBtnFilter.xml Presented in ExcelRT

In ExcelRT, click on Check Box 3 and notice how it gets checked and cell C4 is set to TRUE. From Excel, Right-click on that checkbox and choose **Format Control** from the Pop up menu. Notice the Cell Link field of that control is assigned to C4.

In ExcelRT, click on the radio buttons and notice how the value in cell E6 changes. Those radios buttons have their Cell Link field set to E6.

The Pop up menu and the List Box controls get their data from cells C7:C9. The Pop up menu stores the selected value index at C11 and the List Box stores the selected value index at F11.

Notice how the Spin Button control increments or decrements the value in E10.

Button Actions

Button controls in Excel are often used to run a macro or VBA function. In ExcelRT, a Button control can run one or more script commands or even launch an application written in any programming language.

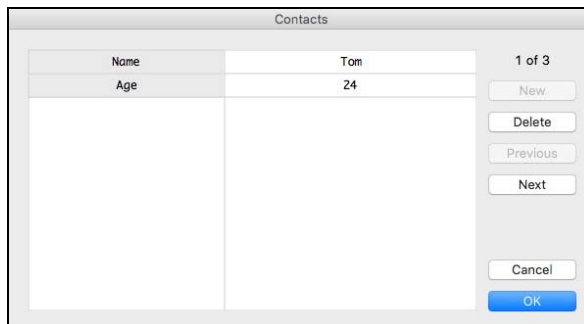
Choose the **Button Actions** command from the **File** menu. Enter the two lines of text shown below and click **OK** to save the dialog. Button Actions are stored in the XML file when you click the **OK** button in the Button Actions dialog. If you save an encrypted file (ERT file), the button actions are compiled into that file.



Button Actions Dialog

The first line assigns `Button 1` to the `DataForm` command. The first parameter of this command assigns the labels used by the columns of data linked to the Data Form dialog. The second parameter assigns the number of rows including the label row. The third parameter provides a title for the Data Form dialog.

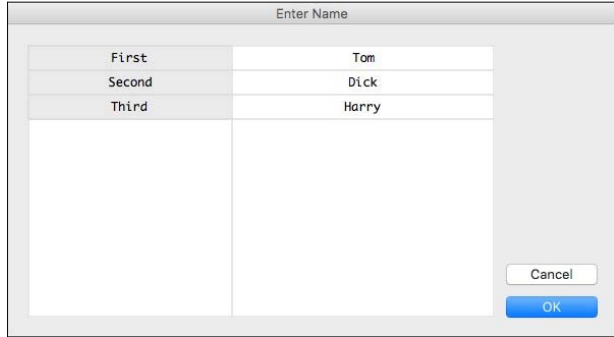
Click `Button 1` and notice the Data Form dialog is presented.



Data Form Dialog

Click `Button 2` to present the Prompt For Value dialog.

This dialog is useful when you want the user to enter a specific set of named values that can be stored as arbitrary cell values.



First	Tom
Second	Dick
Third	Harry

Export Data

To export data in cells `C6:D9` to a text file, present the Button Actions dialog and change the `Button 1` action as illustrated here.

```
Button 1=ExportDataToTicketFolder(C6:D9,COMMA,LF,NQ,"Export.csv")
```

When clicking the button, you'll notice a file named `Export.csv` is created in your shared Ticket folder. The location on that file depends on your OS.

- Mac - `/users/shared/ticket`
- Windows – `c:\users\public\ticket`
- Linux - `/var/ticket`

The export and import script commands allow your workbook to export or import data to files or the clipboard with a variety of formatting options.

Run Application

On Mac, a simple Hello application is included in the Plugins folder within the folder holding ExcelRT. If you double-click that application, you'll notice that it presents a simple Hello window.

On Windows, copy `Hello.exe` from the ExcelRT folder to the Plugins folder. On Linux, you'll need to first set the Execute flag for the executable file using the Properties dialog.

Present the Button Actions dialog and assign `Button 1` to run the application based on the OS you are using. Multiple action commands separated by `|` can be assigned to the same button name.

```
RunMacAppFromPlugin("Hello.app")|RunWinAppFromPlugin("Hello.exe")
```

Now close the dialog and click the button to launch the Hello app.

Send Command from App

An application that you have written can send a command to your workbook running in ExcelRT using either a Clipboard or File command. To enable application commands, present the Button Actions window and set the Enable External Commands checkbox.

The PlainTextEdit application is included with QuickLicense for your convenience when testing Clipboard and File commands. On Linux, it is included with ExcelRT.

Double-click on PlainTextEdit to open an empty text-editing window. Type this simple command into the window.

```
RequestExcelRT:MsgBox("Hello")
```

To simulate a Clipboard command sent from an application to your workbook running in ExcelRT, copy this text into the clipboard. Your workbook presents the Hello message. If the command has a response, its now stored in the clipboard.

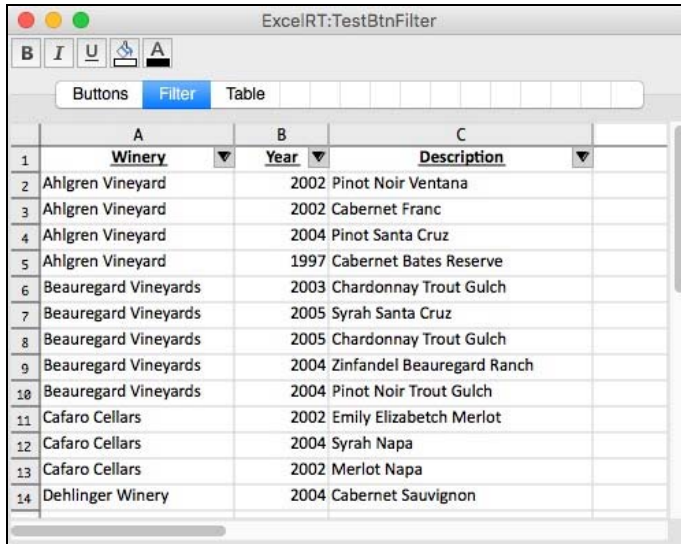
To demonstrate the File method of communicating a command and response with ExcelRT, create a plain text file named ExcelRT.Request. Inside that text file, store just the text of the command, which in this case is:

```
MsgBox("Hello")
```

Now copy and paste file ExcelRT.Request into the shared Ticket folder. ExcelRT presents the Hello message. It also deletes file ExcelRT.Request and writes file ExcelRT.Response into the shared Ticket folder.

Sheet Filter

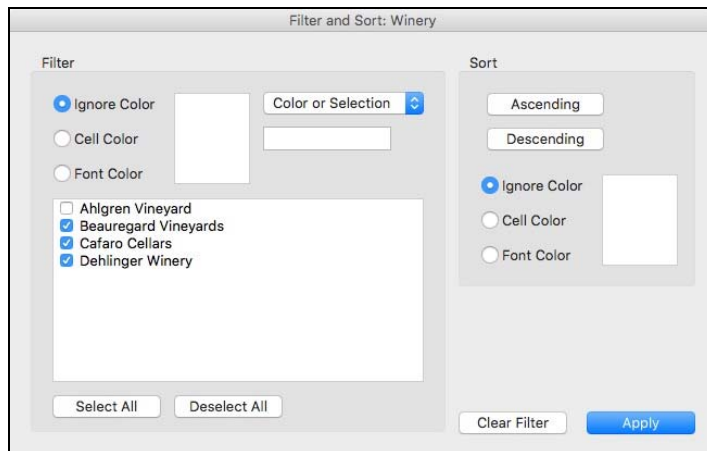
Select the Filter sheet. It contains three columns of data with a Filter icon in the label row of each column.



	A	B	C
1	Winery	Year	Description
2	Ahlgren Vineyard	2002	Pinot Noir Ventana
3	Ahlgren Vineyard	2002	Cabernet Franc
4	Ahlgren Vineyard	2004	Pinot Santa Cruz
5	Ahlgren Vineyard	1997	Cabernet Bates Reserve
6	Beauregard Vineyards	2003	Chardonnay Trout Gulch
7	Beauregard Vineyards	2005	Syrah Santa Cruz
8	Beauregard Vineyards	2005	Chardonnay Trout Gulch
9	Beauregard Vineyards	2004	Zinfandel Beauregard Ranch
10	Beauregard Vineyards	2004	Pinot Noir Trout Gulch
11	Cafaro Cellars	2002	Emily Elizabeth Merlot
12	Cafaro Cellars	2004	Syrah Napa
13	Cafaro Cellars	2002	Merlot Napa
14	Dehlinger Winery	2004	Cabernet Sauvignon

Columns of Data with Sheet Filter

Click the Filter icon in the Winery column to present the Filter and Sort dialog. To hide all rows for the Ahlgren Vineyard, clear that checkbox and click **Apply**.



Filter and Sort: Winery

Filter

☒ Ignore Color ☐ Cell Color ☐ Font Color

Color or Selection

☐ Ahlgren Vineyard
☒ Beauregard Vineyards
☒ Cafaro Cellars
☒ Dehlinger Winery

Select All Deselect All

Sort

Ascending
Descending

☒ Ignore Color ☐ Cell Color ☐ Font Color

Clear Filter Apply

Filter and Sort Dialog for a Sheet or Table Filter

A user can filter and sort rows based on cell or font color. For this to be useful, the user needs an easy way to change cell fill color or font color. When generating an encrypted workbook, you can choose to include a simple ribbon with buttons to change the text style, cell color or font color of the selected cell.

If you would like to experiment with the ribbon features using file TestBtnFilter.xml, quit ExcelRT and edit this file to enable the ribbon. Drag and drop TestBtnFilter.xml onto the PlainTextEdit application.



Edit TestBtnFilter.xml with PlainTextEdit on Windows

At the top of the file, notice the attribute Ribbon="FALSE". Change that to Ribbon="TRUE" and save the file.

Table

Select the sheet named Table. Table rows can be sorted and filtered.

	A	B	C
1	Winery	Year	Description
2	Ahlgren Vineyard	2002	Pinot Noir Ventana
3	Ahlgren Vineyard	2002	Cabernet Franc
4	Ahlgren Vineyard	2004	Pinot Santa Cruz
5	Ahlgren Vineyard	1997	Cabernet Bates Reserve
6	Beauregard Vineyards	2003	Chardonnay Trout Gulch
7	Beauregard Vineyards	2005	Syrah Santa Cruz
8	Beauregard Vineyards	2005	Chardonnay Trout Gulch
9	Beauregard Vineyards	2004	Zinfandel Beauregard Ranch
10	Beauregard Vineyards	2004	Pinot Noir Trout Gulch
11	Cafaro Cellars	2002	Emily Elizabeth Merlot
12	Cafaro Cellars	2004	Syrah Napa
13	Cafaro Cellars	2002	Merlot Napa
14	Dehlinger Winery	2004	Cabernet Sauvignon

Table Presented in ExcelRT