

Application Note

Using VirtualCOM in AutomationDesk

Release 2021-A – May 2021

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	info@dspace.de
Web:	http://www.dspace.com

How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: <http://www.dspace.com/go/locations>
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: <http://www.dspace.com/go/supportrequest>. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit <http://www.dspace.com/go/patches> for software updates and patches.

Important Notice

This publication contains proprietary information that is protected by copyright. All rights are reserved. The publication may be printed for personal or internal use provided all the proprietary markings are retained on all printed copies. In all other cases, the publication must not be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of dSPACE GmbH.

© 2011 - 2021 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

This publication and the contents hereof are subject to change without notice.

AUTERA, ConfigurationDesk, ControlDesk, MicroAutoBox, MicroLabBox, SCALEXIO, SIMPHERA, SYNECT, SystemDesk, TargetLink and VEOS are registered trademarks of dSPACE GmbH in the United States or other countries, or both. Other brand names or product names are trademarks or registered trademarks of their respective companies or organizations.

Contents

About This Document	5
Introduction	7
Intention.....	7
Scope and Area of Validity.....	8
Technical Background	9
AutomationDesk.....	9
Use of COM Objects in AutomationDesk.....	11
Event Handling in AutomationDesk.....	14
Examples	17
Example 1: Sending an Email.....	17
Example 2: Transfer Data between Excel and AutomationDesk.....	19
Example 3: Using Excel Events in AutomationDesk.....	23
Conclusion	29
Comparison of the Assignment Methods.....	29

About This Document





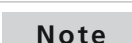


Content This document introduces features that let you access other applications in AutomationDesk sequences via AutomationDesk's VirtualCOM.


Required knowledge Working with AutomationDesk requires:

- Basic knowledge in handling the PC and the Microsoft Windows operating system.
- Basic knowledge in developing applications or tests.
- Basic knowledge in handling the external device, which you control remotely via AutomationDesk.

dSPACE provides trainings for AutomationDesk. For more information, refer to <https://www.dspace.com/go/trainings>.

Symbols dSPACE user documentation uses the following symbols:

Symbol	Description
	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
	Indicates a hazard that, if not avoided, could result in property damage.
	Indicates important information that you should take into account to avoid malfunctions.
	Indicates tips that can make your work easier.
	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.

Symbol	Description
	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

%name% Names enclosed in percent signs refer to environment variables for file and path names.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

Special folders

Some software products use the following special folders:

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>

or

%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>

Documents folder A standard folder for user-specific documents.

%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>

Accessing dSPACE Help and PDF Files


After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as PDF files.

dSPACE Help (local) You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com/go/help.

To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the  icon in dSPACE Help. The PDF opens on the first page.

Introduction

Where to go from here

Information in this section

Intention.....	7
Scope and Area of Validity.....	8

Intention

Introduction

This document is intended for AutomationDesk users who want to use features of other applications in their AutomationDesk sequences. AutomationDesk's open architecture lets users reimplement such features or remote-control other applications.

An interface that is commonly used for remote-controlling software is the Component Object Model (COM). If the application or service you want to remote-control provides a COM interface, you can use it for communication between AutomationDesk and the application. The COM interface allows you to use functions and applications implemented in different programming languages. It can reside on the PC as a dynamic linked library (DLL) or as an executable file (EXE). Features used via the COM interface are not linked or imported into the AutomationDesk project.

The Distributed Component Object Model (DCOM) is based on the same COM technology, except that communication takes place between different network nodes. There are no differences between COM and DCOM with regard to the remote-control interfaces. Other remote-control interfaces like ActiveX or OLE are not discussed in this document.

This document is not an introduction to COM and any remote-controlled application, but a guideline for COM programming via VirtualCOM in AutomationDesk. VirtualCOM is a feature of AutomationDesk and serves as a

protective shield when you use COM in your sequences. It manages COM objects in a secure way to prevent system crashes during the test development phase. This document provides instructions and examples for using COM via VirtualCOM.

Tip

It is recommended to use AutomationDesk's VirtualCOM for COM programming.

Scope and Area of Validity

Introduction

The solutions presented in this document can be used from AutomationDesk 3.2 and upward.

Technical Background

Introduction	This chapter explains the basic technical background of COM and VirtualCOM in AutomationDesk, and introduces recommended implementation structures.
--------------	---

Where to go from here	Information in this section
	AutomationDesk.....9
	Use of COM Objects in AutomationDesk..... 11
	Event Handling in AutomationDesk..... 14

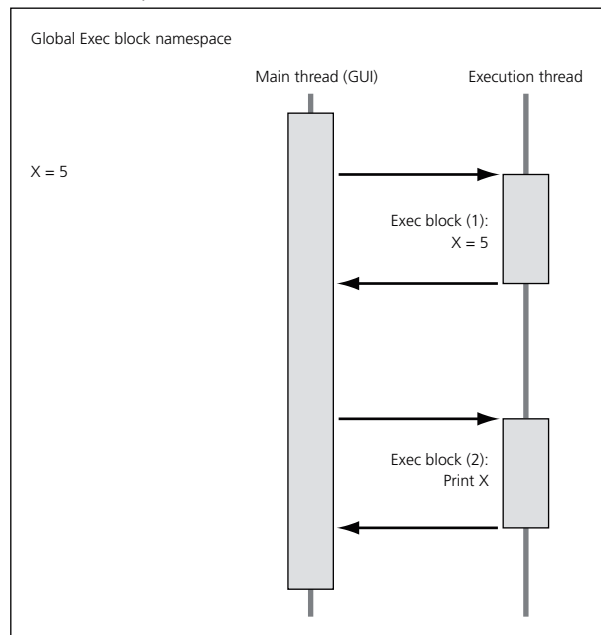
AutomationDesk

Threads in AutomationDesk	<p>The AutomationDesk process contains several threads. The main thread contains the user interface. Each execution of AutomationDesk elements (Project, Folder, Sequence, Serial and Blocks) runs in a separate thread. If subsystems are executed in parallel, each runs in its own thread.</p> <p>Since AutomationDesk 3.2, a further thread, called the <i>container thread</i>, was implemented to manage COM objects created by the VirtualCOM feature.</p>
---------------------------	---

Python namespace in AutomationDesk	<p>AutomationDesk has one global namespace for using Python variables in Exec blocks. The namespace is not cleaned up until the end of the AutomationDesk session. After execution, all assignments made in Exec blocks remain in the namespace and can be accessed in subsequent executions.</p> <p>The figure below shows the global Exec block namespace. In the first execution, Exec block (1) initializes a Python object, an integer variable. In the second</p>
------------------------------------	---

execution, Exec block (2) uses the same namespace and can access the integer variable.

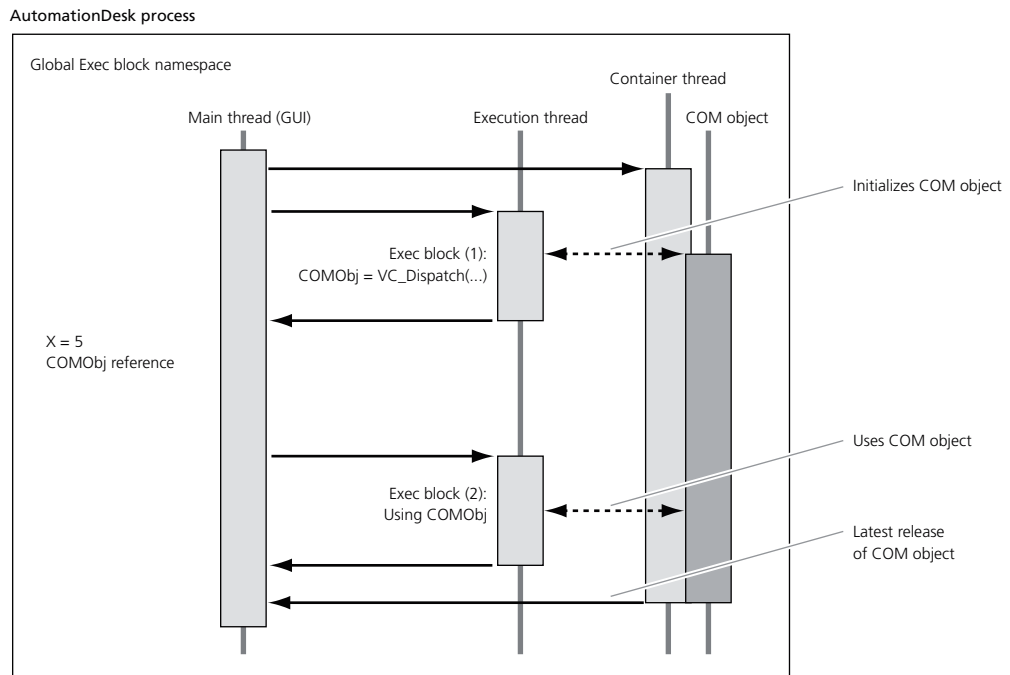
AutomationDesk process



COM objects in AutomationDesk threads

In AutomationDesk each thread has its own message loop. An initialized COM object has to be released before its thread is terminated. With the normal COM object handling, all COM objects have to be released manually by the user's code at the end of each *execution thread*. If VirtualCOM is used to initialize COM objects, they are created in the container thread and are released either manually by the user's code or automatically when AutomationDesk is closed. An object's life time is independent of its execution time.

The figure below shows the global Exec block namespace. In the first execution, Exec block (1) initializes a COM object using the VirtualCOM dispatch method. In the second execution, Exec block (2) is expected to use it. Both the integer object and the COM object are available in the global namespace. The COM object can be used until it is released manually or the container thread is terminated.



Use of COM Objects in AutomationDesk

Introduction

VirtualCOM makes it easy to use COM objects in AutomationDesk. Once a COM object is initialized, it can be used until AutomationDesk is closed. A COM object is not limited to the execution time of a single Exec block, but can be used across blocks, sequences and even folders. The following sections describe how to handle COM objects with VirtualCOM.

Possible methods to assign and clean up COM objects

When using VirtualCOM, you can assign a COM object to a Variant data object in your project structure or to a Python variable.

Note

To use VirtualCOM in your script, you must import the `virtualcomobject` module beforehand.

Using project-specific Variant data objects

If you add a Variant data object to your project, you can access it via the `_AD_` alias.

Assigning COM objects to Variant data objects Because project-specific data objects are elements of the project structure, assigning COM objects to project-specific Variant data objects makes the COM objects visible. This gives

you a list of the COM objects used. Each access to a COM object can be identified by the Variant's name. If COM objects are to be used by several blocks or sequences, it is recommended to use project-specific Variant data objects.

Example: You have added a Variant data object named `ExcelAppl` to your project. To initialize the COM object of the Microsoft Excel application, you can use the following dispatch call.

```
_AD_.ExcelAppl = virtualcomobject.VirtualCOMObject.\
    VC_Dispatch("Excel.Application")
```

Manual cleanup of COM objects using Variant data objects Although the COM objects initialized via VirtualCOM do not have to be cleaned up manually, it may be useful in some cases. They must be released in reverse order of creation.

The Variant data object is initialized automatically with `None` during its creation. After assignment it contains the COM object as its value. To clean up the COM object, just set the Variant data object to `None` again.

```
_AD_.ExcelAppl = None
```

Note

Do not use the `del` command when using Variant data objects for COM objects.

Using Python variables

If you assign a Python variable, it is added to the global namespace and can be accessed by its name.

Assigning COM objects to Python variables The assignment of COM objects to Python variables makes COM objects available in the global namespace for further use. To get the access name of a COM object (the name of the assigned Python variable), you must refer to the block where the object was created. If COM objects are to be used only in a single Exec block, it is recommended to use Python variables.

Example: To assign the COM object of the Microsoft Excel application to the Python variable named `ExcelApp`, you can use the following dispatch call.

```
ExcelApp = virtualcomobject.VirtualCOMObject.\
    VC_Dispatch("Excel.Application")
```

Manual cleanup of COM objects using Python variables Although the COM objects initialized via VirtualCOM do not have to be cleaned up manually, it may be useful in some cases. They must be released in reverse order of creation.

The following program listing shows how to initialize, use and clean up COM objects. Initialization is necessary because an error can occur before a COM object is created. In this case the `del` statement itself would raise an error and subsequent code lines in the cleanup part would not be executed.

```

# Initialize the COM objects
ComObj_1 = None
ComObj_2 = None

# Creating and using COM objects
try:
    <...>

# Releasing the COM objects
finally:
    del ComObj_2
    del ComObj_1

```

Example of using Python variables For example, Microsoft Excel provides an application object with workbook objects under it in the hierarchy, and objects for single Excel sheets at the lowest level. Theoretically, the server knows the cleanup order of all objects and shuts down after releasing the last COM object independently of its hierarchy level. The server documentation should also explain the rules for shutdown.

However, it is more secure to clean up the COM objects in reverse order of hierarchy level or creation:

```

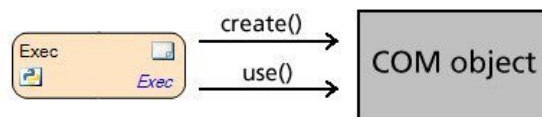
# Creation order
ExcelApp1 = virtualcomobject.VirtualCOMObject.\
    VC_Dispatch("Excel.Application")
Workbook = ExcelApp1.Workbook.Item(...)
Sheet = Workbook.Sheet.Item(...)
# Cleanup order (inverse to the creation order)
del Sheet
del Workbook
del ExcelApp1

```

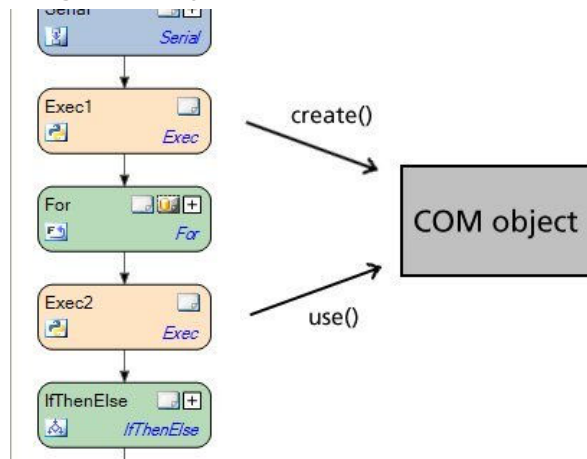
Using COM objects initialized by VirtualCOM

COM objects are not limited by the implementation structure. They can be used in a single Exec block, across several Exec blocks within a sequence, across several sequences and even across different folders, as shown in the illustrations below.

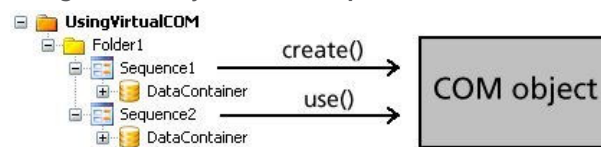
Using a COM object in a single Exec block



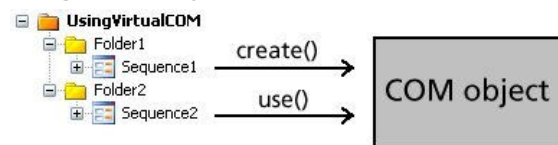
Using a COM object across blocks



Using a COM object across sequences



Using a COM object across folders



Event Handling in AutomationDesk

Introduction

With AutomationDesk 3.4, the VirtualCOM feature also provides COM-based event handling.

AutomationDesk can react on events provided by the connected COM application. After execution, the defined application event classes have to be cleared.

Defining an application event class

Before AutomationDesk can react to events of the connected COM application, you have to define a class in AutomationDesk containing the relevant methods.

Refer to the COM API documentation of your COM application to get the names of the available methods and parameters.

Example

```
class AppEvents:
    # You can use any name for the class definition
    def Method1(self, Parameter1):
        print("*Event* APP -> Method1:", Parameter1)
    def Method2(self, Parameter1, Parameter2):
        print("*Event* APP -> Method2:", Parameter1, Parameter2)
```

Note

There might be methods that are only available in the main thread of the COM object. Then, AutomationDesk cannot access such an event.

Connecting to the events

After the application event class is defined and the application object is created, you have to connect the application events to the application object by using the `VC_DispatchWithEvents` method.

The syntax of the method is:

```
VC_DispatchWithEvents(VC_ApplObject, ApplEventClass)
```

Example

```
_AD_.AppEvents = virtualcomobject.VirtualCOMObject.\
VC_DispatchWithEvents(_AD_.MyApp1, AppEvents)
```

Using the events

While the application object and the application event class is connected, AutomationDesk can react to the specified actions in the COM application.

Disconnecting from the events

Before you clear the application object, you should disconnect and clear the application event class. You do this by setting the application event class to `None`.

Example

```
_AD_.AppEvents = None
```

Example project

For an example project, refer to [Example 3: Using Excel Events in AutomationDesk](#) on page 23.

Examples

Introduction

This chapter shows the implementation structure for COM handling using the two types of COM object assignment. The first example shows a remote control for Microsoft Outlook®. The implementation is based on Python variable assignment, and all the COM handling is implemented in a single Exec block.

Data object assignment is described in the second example, a remote control for Microsoft Excel®. The COM handling is distributed over several Exec blocks and sequences.

Where to go from here

Information in this section

[Example 1: Sending an Email..... 17](#)

This example is implemented in a single Exec block and the COM objects are assigned to Python variables.

[Example 2: Transfer Data between Excel and AutomationDesk..... 19](#)

This example is implemented in two sequences. The COM objects are assigned to project-specific Variant data objects.

[Example 3: Using Excel Events in AutomationDesk..... 23](#)

This example shows you how to access events from a remote-controlled Excel application.

Example 1: Sending an Email

Introduction

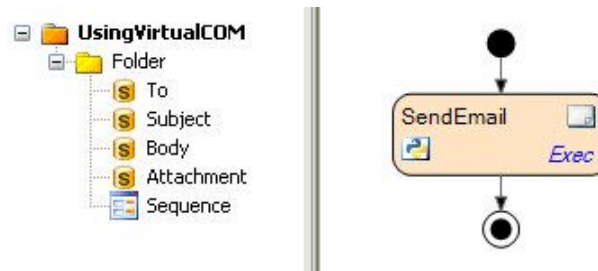
This example is implemented in a single Exec block, and the COM objects are assigned to Python variables.

It shows how you can implement a remote control for Microsoft Outlook® to send an e-mail.

Implementation of Example 1

Block interface and Python code The Python code contains the following variables that are to be provided by project-specific data objects:

Name	Data Type	Description
To	String	Addresses of the e-mail
Subject	String	Subject of the e-mail
Body	String	E-mail text
Attachment	String	Path of the attachment file



The Python code of the Exec block to remote-control Outlook contains two COM objects, the application object (Python variable Outlook) and the object of the email (Python variable EMail). The email object depends on the application object and needs to be released before the application object is released.

```
import virtualcomobject

Outlook = None
EMail = None

try:
    Outlook = virtualcomobject.VirtualCOMObject.\
        VC_Dispatch("Outlook.Application")
    if Outlook is not None:
        EMail = Outlook.CreateItem(0)
        if EMail is not None:
            EMail.To = _AD_.To
            EMail.Subject = _AD_.Subject
            EMail.Body = _AD_.Body
            if _AD_.Attachment is not '':
                EMail.Attachments.Add(_AD_.Attachment,1,1)
            EMail.Send()
finally:
    del EMail
    del Outlook
```

Behavior of Microsoft Outlook during Remote Control The behavior of Outlook during remote control is as follows: If Outlook is closed, COM object creation opens it, and the final COM object cleanup closes it again. If Outlook is already open, the new COM object connects to it, and the clean up only closes the connection between the COM object and Outlook.

Example 2: Transfer Data between Excel and AutomationDesk

Introduction

This example is implemented in two sequences. The COM objects are assigned to project-specific Variant data objects.

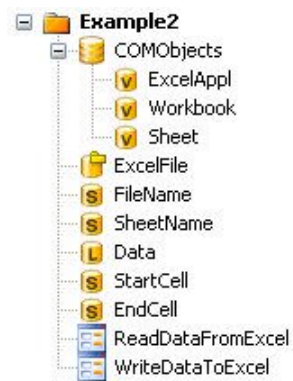
The aim is to remote control the data transfer between Microsoft Excel® and AutomationDesk. In the first sequence, data is read from an Excel file, and in the second sequence, data is written to the Excel file. Using VirtualCOM, the initialized COM objects for reading data in the first sequence can also be used for writing data in the second sequence. Because different Exec blocks have to access the COM objects, they are made visible as project-specific Variant data objects in the project structure.

Implementation of Example 2

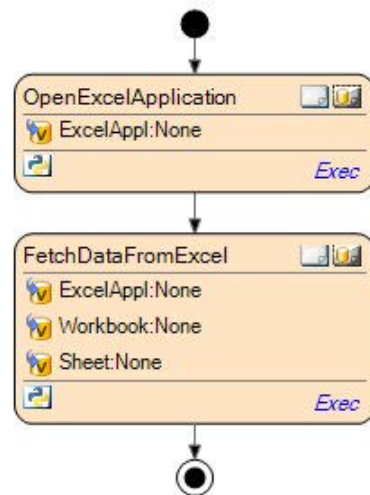
Behavior of Microsoft Excel during remote control The Excel application is used multiple times during project execution. Hence, it is better to open Excel at the beginning of the execution and not to close it until the end of the execution. The read and write access is realized in several separate Exec blocks using the two sequences `ReadDataFromExcel` and `WriteDataToExcel`.

Block Interfaces

Name	Data Type	Description
COMObjects	DataContainer	Contains COM objects.
ExcelApp	Variant	Used for the COM object of the Excel application.
Workbook	Variant	Used for the COM object of the Excel workbook.
Sheet	Variant	Used for the COM object of the Excel sheet.
ExcelFile	File	Path of the Excel file
FileName	String	Name of the Excel file
SheetName	String	Name of the Excel sheet
Data	List	List for storing the data from the Excel sheet in the form [[row][row]]
StartCell	String	First cell of the data in the Excel sheet, for example, A2
EndCell	String	Last cell of the data in the Excel sheet, for example, E6



ReadDataFromExcel sequence The ReadDataFromExcel sequence contains the two Exec blocks OpenExcelApplication and FetchDataFromExcel.



Implementation of the OpenExcelApplication block This Exec block provides a Variant data object that references the ExcelAppl data object of the project. If you execute the block, the Excel application COM object is assigned to the project-specific Variant data object in the global namespace of AutomationDesk. Other Exec blocks can use this global data object to access Excel. After assignment, Excel is started.

```
import virtualcomobject

_AD_.ExcelAppl = virtualcomobject.VirtualCOMObject.\
    VC_Dispatch("Excel.Application")
_AD_.ExcelAppl.Visible = 1
```

Implementation of the FetchDataFromExcel block This Exec block provides three Variant data objects which reference the corresponding data objects ExcelAppl, Workbook and Sheet in the project. These data objects are in the global namespace of AutomationDesk and can be used directly in this Exec block without creating additional data objects.

Tip

Although it is not mandatory to add the ExcelAppl data object to the block's interface, it is recommended to do so for a better overview of the COM objects used without looking into the block's code.

If you execute the block, you get the data from the Excel sheet and store it in the Data data object.

```
# Open the Excel file
_AD_.ExcelAppl.Workbooks.Open(_AD_.ExcelFile.GetPathName())

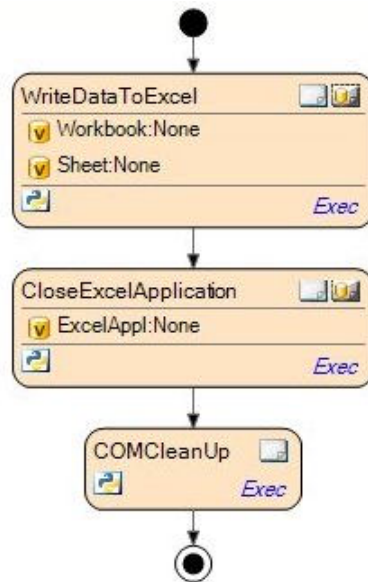
# Fetch COM interface of the Excel workbook
_AD_.Workbook = _AD_.ExcelAppl.Workbooks.Item(_AD_.FileName)

# Fetch COM interface of the Excel sheet
_AD_.Sheet = _AD_.Workbook.Worksheets.Item(_AD_.SheetName)

# Fetch the sheet from the Excel file. Return value is a tuple.
Data = _AD_.Sheet.Cells.\
    Range(_AD_.StartCell,_AD_.EndCell).Value

# Data : tuple of tuples -> convert to list of lists
_AD_.Data = map(list, Data)
```

WriteDataToExcel sequence The WriteDataToExcel sequence contains the three Exec blocks WriteDataToExcel, CloseExcelApplication and COMCleanUp.



Implementation of the WriteDataToExcel block This Exec block provides the two data objects Workbook and Sheet to show that it is using the corresponding COM objects.

If you execute the block, the data fetched from the Excel file by the first sequence is written back to the Excel sheet as a copy. Finally the Excel file is saved.

```
# Write the data back to the Excel sheet
_AD_.Sheet.Cells.Range("A11", "E15").Value = _AD_.Data

# Save the Excel file
_AD_.Workbook.Save()
```

Implementation of the CloseExcelApplication block This Exec block provides the data object ExcelAppl to show that the corresponding COM object is used in the block's code. If you execute the block, the Excel application is hidden and closes.

```
_AD_.ExcelAppl.Visible = 0
_AD_.ExcelAppl.Quit()
```

Implementation of the COMCleanUp block This Exec block cleans up the three COM objects Sheet, Workbook and ExcelAppl by setting them to None. The COM objects must be released in reverse order of creation. The Sheet object is released first, then the Workbook object and finally the ExcelAppl object. This block shows that no data objects are required in the block's interface to access the corresponding global data objects. If you execute the block, the COM objects that are represented by the corresponding Variant data objects in the COMObjects data container are cleaned up by setting them to None.

```
_AD_.COMObjects.Sheet = None
_AD_.COMObjects.Workbook = None
_AD_.COMObjects.ExcelAppl = None
```

The COM objects are automatically cleaned up when you close AutomationDesk.

Example 3: Using Excel Events in AutomationDesk

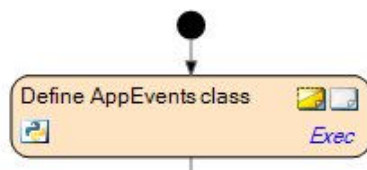
Introduction This example shows you how to access events from a remote-controlled Microsoft Excel® application in AutomationDesk.

Implementation of Example 3 **Project-specific data objects** The following Variant data objects are used to provide the created COM objects.

Name	Data Type	Description
ExcelAppl	Variant	Used for the COM object of the Excel application.
ExcelEvents	Variant	Used for the COM object of the connected application event class.



Define application event class The GetEventsFromExcel sequence starts with an Exec block to define the application event class.



It contains the following script:

```
class AppEvents:
    def OnWindowResize(self, Wb, Wn):
        print("*Event* APP -> Resize:", Wb, Wn)
        # This is an example of a method that cannot
        # be accessed via AutomationDesk
    def OnNewWorkbook(self, Wb):
        print("*Event* APP -> NewWorkbook:", Wb)
```

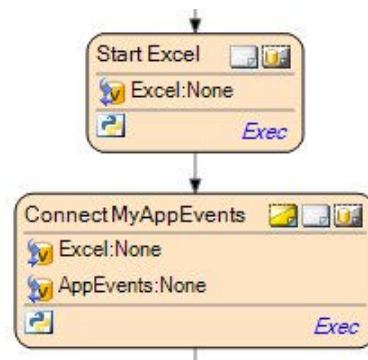
Connect the application event class to the application object The application event class has to be connected to the formerly created application object. To simplify the access to the project-specific ExcelAppl data object, each block that needs access to the Excel application object has an internal Excel data object that refers to it.

For creating the Excel application object, type the following script in the Start Excel block. The block returns the Excel application object and starts Excel.

```
import virtualcomobject
_AD_.Excel = virtualcomobject.VirtualCOMObject.\
    VC_Dispatch("Excel.Application")
```

The Connect MyAppEvents block provides two data objects: the Excel data object to refer to the project-specific application object, and the AppEvents data object, to refer to the project-specific ExcelEvents event object. The block reads the application object and returns the created event object. Then, the application event class is connected to the Excel application.

```
import virtualcomobject
_AD_.AppEvents = virtualcomobject.VirtualCOMObject.\
    VC_DispatchWithEvents(_AD_.Excel, AppEvents)
```

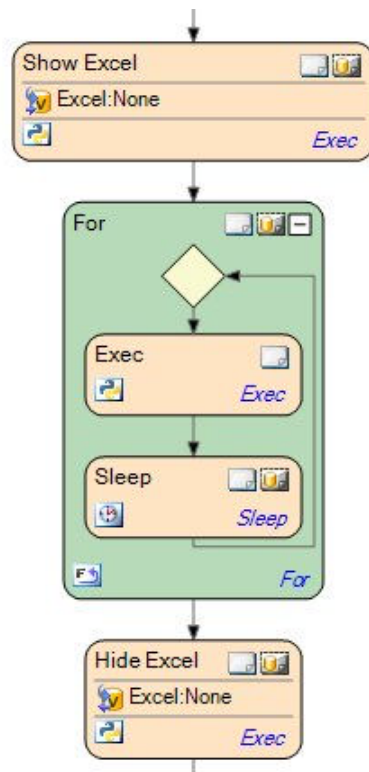



Using the application event class During the For loop, you can create a new workbook and resize the Excel window. The creation of the workbook triggers the specified event and you can see the message defined in the application event class in the Output Viewer. The resize of the window also triggers an event but it cannot be accessed by AutomationDesk. The defined message is not displayed. While you want to work with the Excel user interface, the application must be visible.

To change the visibility of Excel, you can use the Visible property:

```

_AD_.Excel.Visible = True    # user interface is displayed
_AD_.Excel.Visible = False   # user interface is hidden
  
```



Cleaning up Excel The clean up process includes three steps.

1. Clear the application event class

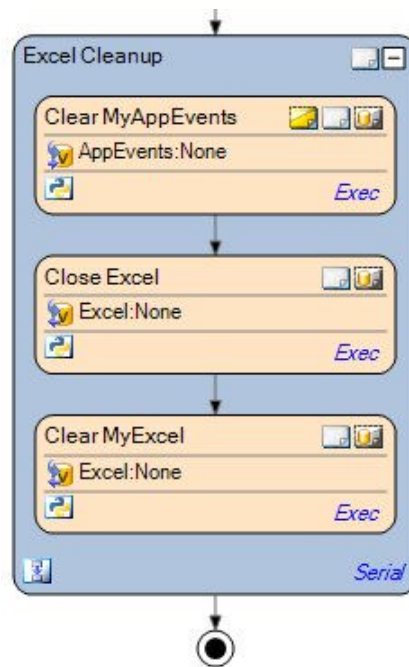
```
_AD_.AppEvents = None
```

2. Close Excel

```
_AD_.ExcelAppl.Quit()
```

3. Clear the Excel application object

```
_AD_.Excel = None
```



If you execute the blocks, the application event class is disconnected from the application object and cleared, Excel is closed, and the Excel application object is cleared.

Conclusion

Introduction

The descriptions and examples in this document demonstrate the functionality of AutomationDesk's VirtualCOM. Its main advantage over the standard COM handling is easier use of COM objects. There are no strict rules you have to observe to prevent run-time errors, for example, when using the Stop Execution command. VirtualCOM in AutomationDesk is a convenient, secure way to handle COM objects for remote control.

Comparison of the Assignment Methods

Assignment methods

The following table shows the recommended COM handling for the two methods of COM object assignment.

	Using Python Variable	Using Variant Data Object
COM objects used in one single Exec block	... in several Exec blocks, sequences or folders
Initialize and manual cleanup	<ol style="list-style-type: none">1. Initialize variable with None before assigning to the COM object2. Clean up code: <code>del variable</code>	<ol style="list-style-type: none">1. Add a Variant data object to the project.2. Clean up code: <code>_AD_.VariantName = None</code>

