ControlDesk

Customization

For ControlDesk 7.4

Release 2021-A – May 2021

dSPACE

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

Contents

About This Document

Content

This document introduces you to customizing ControlDesk.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	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.
	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.

Tutorial Videos

Tutorial Videos

Introduction	The dSPACE website provides customization tutorial videos.
Customizing ControlDesk instruments	You can add a Python automation script to an instrument to customize it and extend its functionality. The tutorial videos show you: ▪ How to add a Python script to an instrument and make the instrument a custom instrument. ▪ How to extend instrument functionality via instrument scripts. The stopwatches in the Instrument Selector are used as examples. Refer to https://www.dspace.com/go/tutorial_cd_custom_instr.
Public product videos	For public product videos, refer to ControlDesk product videos.
Related topics	Basics Tutorial Videos for ControlDesk (ControlDesk Introduction and Overview 🕮)

Basics and Instructions

Where to go from here

Information in this section

Creating a Project When ControlDesk Starts Up

Creating a Project When ControlDesk Starts Up

Introduction

You can use ControlDesk events to create a project automatically when ControlDesk starts up.

Started event

You can use the `Started` application event in the user event context, for example, to create a project automatically when ControlDesk starts up.

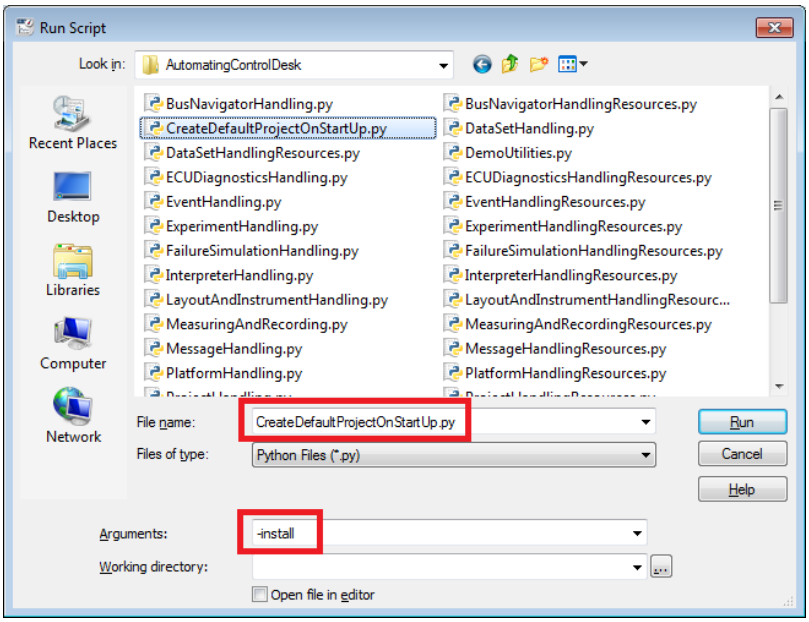
Demo script

For demonstration purposes, ControlDesk provides the CreateDefaultProjectOnStartUp demo. The demo uses the `Started` event to create a default project and experiment automatically when ControlDesk starts up. If there are registered platforms, these are added to the experiment.

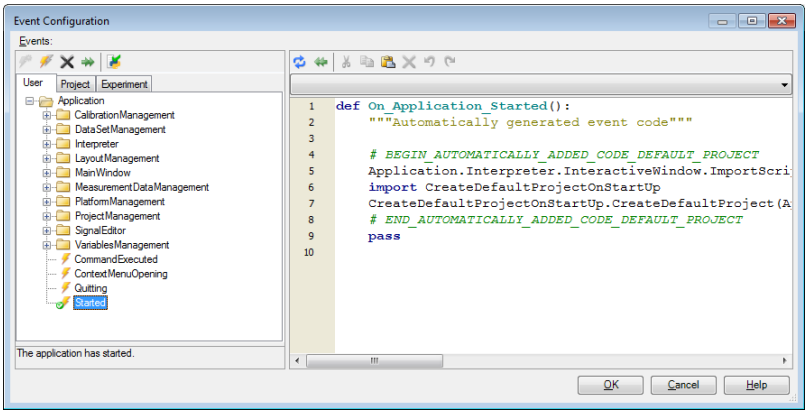
You can adapt the demo to meet your requirements.

Using the demo to create a project when ControlDesk starts up To create a project automatically when ControlDesk starts up, perform the steps below:

1. In ControlDesk, run the script with the `-install` argument.



The `-install` argument inserts the code for project creation in the user event context of ControlDesk's event configuration as a new event handler. The event handler is activated.



2. Close ControlDesk.
3. Restart ControlDesk.

   When ControlDesk starts up, a default project is created.

Tip

As an alternative, you can execute the demo script as extension script. Extension scripts are executed automatically when ControlDesk starts up. Refer to the *tips/remark* section in the demo script. For detailed information on extension scripts, refer to Executing Extension Scripts When ControlDesk Starts Up on page 12.

Location of the demo The CreateDefaultProjectOnStartUp demo is located in the `.\Demos\ToolAutomation\<ProgrammingLanguage>\` folder of your ControlDesk installation.

Related topics

Basics

Executing Extension Scripts When ControlDesk Starts Up.....12

References

ApplicationEvents / IXaApplicationEvents <<EventInterface>> (ControlDesk Automation 🕮)

Executing Extension Scripts When ControlDesk Starts Up

Executing Extension Scripts When ControlDesk Starts Up

Introduction

You can specify Python scripts as *extension scripts*. These scripts are executed automatically when ControlDesk starts up.

Extension scripts

Basics Extension scripts are Python scripts (PY or PYC files) that are executed each time ControlDesk starts up. They share the following specifics:

- Extension scripts are executed in a common Python interpreter that is independent of ControlDesk's internal **Interpreter**.

  Note

  The common interpreter ignores Python search paths specified on the **Interpreter** page. As an alternative, specify additional search paths directly in the Python installation to make them accessible to the common interpreter.

  Note

  As of ControlDesk 5.4, Python-specific environment variables such as `PYTHONHOME` are no longer evaluated by ControlDesk's **Internal Interpreter** and when they are used in extension scripts.

- Extension scripts access ControlDesk's automation interface via a global variable named `Application`.
- Extension scripts are specified via XML configuration files.
- Extension scripts can be executed for all users or user-specifically.

Tip

You can use extension scripts, for example, to customize ControlDesk's ribbon by adding your own ribbon controls. For details, refer to Basics on Customizing Ribbons via Extension Scripts on page 16.

Namespace and module import

- Each extension script has its own namespace.

  If an error occurs when a script is started, the namespace is deleted.
- When you import *global modules and/or packages* located in the Python search path, these modules and/or packages are loaded only once, i.e., when the first extension script importing these modules/packages is executed. All the other extension scripts share the loaded modules/packages.

- In addition, you can also import *local modules* located in the extension script folder. These local modules are loaded only to the related extension script.

Extension script execution All extension scripts are executed immediately after the `Started` application event. The `Started` event itself is not available for the extension script.

Extension scripts are executed in a common Python interpreter that is independent of the Internal Interpreter.

Path and arguments

- `sys.argv[0]` contains the path of the script.
- `sys.argv[1:]` contains all the arguments specified in the configuration file.

`__name__` variable The `__name__` variable has the value `__extension__`.

Print output Print output is stored as an info message in the `dSPACE.log` file.

- To print extension script outputs to the **Interpreter** controlbar, you have to use the automation interface of the Internal Interpreter.

  The following listing shows an example:

  ```
  Application.Interpreter.InteractiveWindow.Exec("print('Hello World.')")
  ```

  Refer to `Interpreter / IPiInterpreter <<Interface>>` (ControlDesk Automation 🕮).
- To print extension script outputs to the **Messages controlbar**, you have to use its automation interface.

  The following listing shows an example:

  ```
  Application.Log.WriteInformation("Hello World.")
  ```

  Refer to `Log / ILoLog <<Interface>>` (ControlDesk Automation 🕮).

Error output Error output is displayed in the **Messages controlbar**.

XML configuration files

You have to specify extension scripts in an XML configuration file, which must have the `EXTSCRIPT` file name extension.

Configuring the execution of extension scripts A configuration file lets you specify the execution details of one or more extension scripts:

- You can specify the names and relative paths of extension scripts.
- You can enable and disable the execution of an extension script.
- You can add command line parameters to an extension script.
- If you use extension scripts for ribbon customization, the XML configuration file must also contain a link to the custom UI extension file specifying the ribbon customization.

Location of configuration files Configuration files must be stored in an `ExtensionScripts` folder.

ControlDesk offers two locations in the file system where you can add an `ExtensionScripts` folder. Depending on the location, an extension script is executed for a specific user or for all users:

- *For a specific user*

  Scripts that are to be executed for a specific user must be specified in a configuration file stored in the `<DocumentsFolder>/ExtensionScripts` folder.

  For the location of the *Documents folder*, refer toDocuments folder.
- *For all users*

  Scripts that are to be executed for all users must be specified in a configuration file stored in the `<CommonProgramDataFolder>/ExtensionScripts` folder.

  For the location of the *Common Program Data folder*, refer to Common Program Data folder.

Tip

To get, for example, the `<DocumentsFolder>` location via automation, you can use the properties of the `ApplicationEnvironment / IAeApplicationEnvironment <<Interface>>` interface.
Refer to ApplicationEnvironment / IAeApplicationEnvironment <<Interface>> (ControlDesk Automation).

Processing configuration files ControlDesk processes all the configuration files stored in the `ExtensionScripts` folders. The scripts for specific users are processed first.

Example of a configuration file

The following configuration file specifies the execution details of three extension scripts.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ExtensionScripts>
 <PythonScript RelativePath="menu_extension.py" />
 <PythonScript RelativePath="Foo\foo.pyc"/>
  <Arguments>
   <Argument>-d</Argument>
   <Argument>data</Argument>
  </Arguments>
 </PythonScript>
 <PythonScript RelativePath="toto.py" Enabled="false" />
</ExtensionScripts>
```

The configuration file contains the following specifications:

- The Python script `menu_extension.py` is stored in the same folder as the configuration file. This script is loaded when ControlDesk starts up.
- The Python script `foo.py` is stored in the `Foo` subfolder. The command line arguments `-d data` are passed to this script when it is loaded.
- The Python script `toto.py` is not loaded at the ControlDesk start, because `Enabled` is set to `False`

Example of an extension script

The following extension script adds the **Show Item Names** command to the context menu of a selected item in ControlDesk, for example, an instrument on a layout. When you select the new command, an info message with the item's name is stored in the **dSPACE LOG** file.

```
from win32com.client import Dispatch, DispatchWithEvents
class ApplicationEvents(object):
    """Handle Application events."""
    def OnContextMenuOpening(self, Menu, Selection):
        """Called when an extensible context menu is opened."""
        Menu = Dispatch(Menu)
        # Add 'Show Item Names' command is selection is not empty.
        if len(Selection):
            Menu.AddSeparator()
            Command = Menu.AddCommand()
            Command.Key = "ShowItemNames"
            Command.Caption = "_Show Item Names"
    def OnCommandExecuted(self, Command, Selection):
        """Called when the user executed a command created by menu extension."""
        Command = Dispatch(Command)
        if Command.Key == "ShowItemNames":
            # Handle 'Show Item Names' command.
            print ">>> Item Names:"
            for Item in Selection:
                Item = Dispatch(Item)
                if hasattr(Item, "Name"):
                    print "      %s" % Item.Name
                elif hasattr(Item, "FileName"):
                    print "      %s" % Item.FileName
if __name__ == "__extension__":
    Application = DispatchWithEvents(Application, ApplicationEvents)
elif __name__ == "__main__":
    Application = DispatchWithEvents("ControlDeskNG.Application", ApplicationEvents)
    Application.MainWindow.Visible = True
```

The first **if** statement is used if the script is executed as an extension script, the second is used if it is executed in an external browser for test purposes.

Related topics

Basics

Customizing the Ribbon via Extension Scripts 16

Customizing the Ribbon via Extension Scripts

Where to go from here

Information in this section

Basics on Customizing Ribbons via Extension Scripts

Introduction

You can customize ControlDesk's ribbon by adding custom ribbon controls. You have to link each custom ribbon control to a Python *extension script*, which allows you to add custom functions based on ControlDesk's automation interface to the user interface.

Basics on ribbon customization

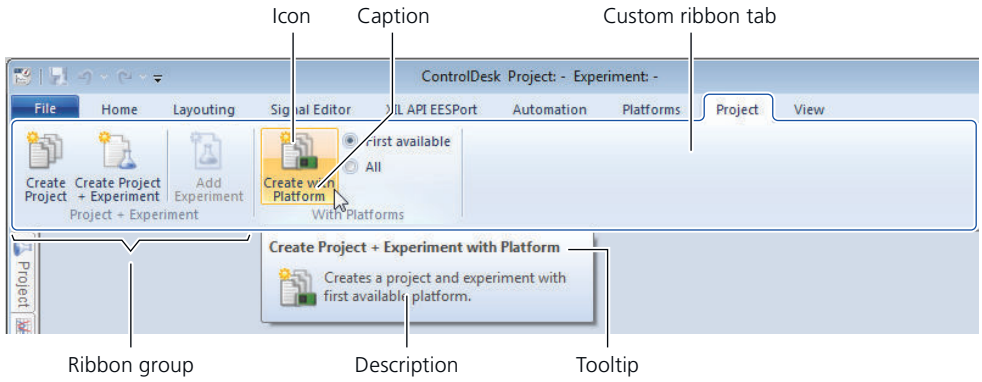
You can perform the following ribbon customization:

- Insert ribbon tabs at specific positions on the ribbon.
- Add new ribbon groups to a ribbon tab.
- Add the following ribbon controls to a custom ribbon group and customize their appearance:
  - Buttons
  - Toggle buttons
  - Checkboxes
  - Radio buttons
  - Split buttons and menu buttons (providing a menu with ribbon controls)
  - Labels (displaying static text)
  - Separators (separating ribbon controls in a ribbon group)

  Custom ribbon controls can only be added to custom ribbon groups.

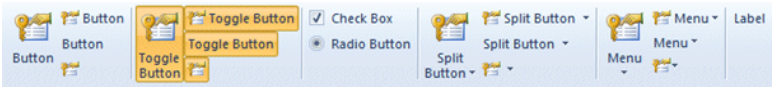
  Each custom ribbon control must be linked to a Python *extension script* that implements one or more custom functions.

As an example, the illustration below shows the **Project** custom ribbon tab inserted before the **View** ribbon tab. The **Project** tab has two custom ribbon groups, each of which contains various ribbon controls.



Ribbon controls

The illustration below shows all the available ribbon controls and their display style:

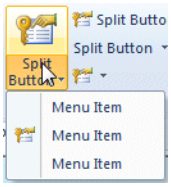


- Buttons and toggle buttons can have icons. The toggle buttons in the illustration above are in the 'on' state.
- Check boxes and radio buttons cannot have icons.
- Split buttons and menu buttons have submenus.

  A split button consists of two elements:
  - The upper element directly invokes a function.

  - The lower element opens a submenu.

Customization via extension scripts

The custom function to be carried out when the user invokes a custom ribbon control usually is implemented in a Python *extension script*. Extension scripts are executed automatically when ControlDesk starts up and have full access to ControlDesk's automation interface.

To perform ribbon customization, you have to provide the following files:

- An extension script (PY or PYC file) implementing the custom function
- A custom UI extension XML file specifying the ribbon extension

  For a description of the schema of the custom UI extension file, refer to Schema of the Custom UI Extension File on page 21.
- An XML configuration file with the `EXTSCRIPT` file name extension

The configuration file must reference the extension script and the custom UI extension file. Refer to Executing Extension Scripts When ControlDesk Starts Up on page 12.

For an example, refer to Example of Customizing Ribbons via Extension Scripts on page 18.

Making ribbon tabs invisible/visible

The `SetTabVisibility` method of the `Ribbon / IUiRibbon <<Interface>>` lets you make ribbon tabs invisible/visible. When you use this method, you have to specify the unique key of the ribbon tab as defined in Built-in tabs and their keys on page 22.

Example The following listing shows how to make the ControlDesk Signal Editor ribbon invisible as an example:

```
Application.MainWindow.Ribbon.SetTabVisibility("ControlDeskNG.TabSignalEditor",False)
```

Refer to Ribbon / IUiRibbon <<Interface>> (ControlDesk Automation 🕮).

Related topics

Basics

Executing Extension Scripts When ControlDesk Starts Up..... 12

Examples

Example of Customizing Ribbons via Extension Scripts..... 18

References

Schema of the Custom UI Extension File..... 21

Example of Customizing Ribbons via Extension Scripts

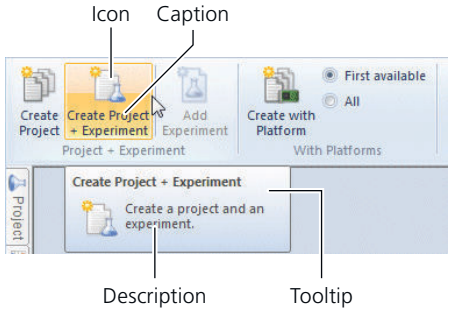
Example overview

What you will learn The example shows how to:

- Insert the Project custom ribbon tab before the View built-in ribbon tab
- Add the Project + Experiment ribbon group to the Project ribbon tab
- Add the Create Project + Experiment button to the Project + Experiment ribbon group
- Make the following specifications for the Create Project + Experiment button:
  - A caption
  - A tooltip
  - A description

- A keytip
- An icon

See the illustration below:



Example based on CustomUI demo The example described below is based on the *RibbonTabExample* extension script of the CustomUI tool automation demo.

To extend the ControlDesk ribbon by using this demo, perform the following steps:

1. From the `.\Demos\ToolAutomation\Python\CustomUI` folder of your ControlDesk installation, copy the `RibbonTabExample.extscript` file and the entire `RibbonTabExample` folder to the following folder:

   `%USERPROFILE%\Documents\dSPACE\ControlDesk\7.4\ExtensionScripts`
2. Restart ControlDesk.

Tip

As an alternative, you can use an installation script that installs both examples of the CustomUI demo (*RibbonTabExample* and *ContextMenuExample*) in one step. Refer to the `info.txt` file in the `CustomUI` demo folder.

Extension script example

The listing below shows excerpts from the `RibbonTabExample.py` file. When the `RibbonTabExample.ButtonProjectExperiment` command is selected, a project and an experiment are created.

```
from win32com.client import DispatchWithEvents
class ApplicationEvents(object):
   """Defines the event sink for the application events.
   def OnCommandExecuted(self, Command, Selection):
        """This method will be called if a command has been executed by the user.
        Syntax      : Obj.OnCommandExecuted()
        Parameters  : Command   - object - The command that has been executed.
                      Selection - object - The currently selected objects.
        Description : This method will be called if a command has been executed by the user.
```

```
        Return Value: -
        """
        Command = Dispatch(Command)
        if Command.Key == "RibbonTabExample.ButtonProjectExperiment":
            AddProject()
            AddExperiment()
        elif (...)
    def AddProject():
        (...)
    def AddExperiment():
        (...)
if __name__ == "__extension__":
        # Advise event class.
        ApplicationWithEventSink = DispatchWithEvents(Application, ApplicationEvents)
        (...)
```

Custom UI extension file example

The listing below shows an excerpt from the `RibbonTabExample.xml` custom UI extension file specifying the ribbon extension:

```
<?xml version="1.0" encoding="UTF-8"?>
<customUI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="CustomUIExtensions.xsd">
   <ribbon>
      <tabs>
         <tab key="RibbonTabExample.TabProject" caption="Project" keytip="J" insertBefore="ControlDeskNG.TabView">
            <group key="RibbonTabExample.GroupProject" caption="Project + Experiment">
               (...)
               <button key="RibbonTabExample.ButtonProjectExperiment" caption="Create Project + Experiment"
                       tooltip="Create Project + Experiment" description="Create a project and an experiment."
                       keytip="E" icon="ProjectExperiment.png" style="bigicontext"/>
               (...)
            </group>
            (...)
         </tab>
      </tabs>
   </ribbon>
</customUI>
```

EXTSCRIPT XML configuration file example

The listing below shows the XML configuration file that references the extension script and the `RibbonTabExample.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<ExtensionScripts xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="ExtensionScripts.xsd">
    <PythonScript RelativePath="RibbonTabExample\RibbonTabExample.py" />
    <CustomUI RelativePath="RibbonTabExample\RibbonTabExample.xml">
</ExtensionScripts>
```

Related topics

Basics

Basics on Customizing Ribbons via Extension Scripts..... 16
Executing Extension Scripts When ControlDesk Starts Up..... 12

Tool Automation Demos (ControlDesk Introduction and Overview 🕮)

References

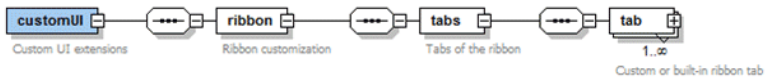
Schema of the Custom UI Extension File

Introduction

Describes the schema of the custom UI extension file, which is an XML file specifying a custom ribbon extension.

customUI root element

The custom UI extension file must contain the `customUI` root element with one `ribbon` element. The `ribbon` element must contain a `tabs` element with at least one `tab` element.



Structure of a custom UI extension file The structure of a custom UI extension file therefore looks like this:

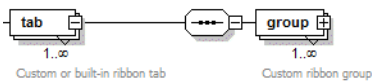
```
<?xml version="1.0" encoding="UTF-8"?>
<customUI>
  <ribbon>
    <tabs>
      …
    </tabs>
  </ribbon>
</customUI>
```

tab element

The custom UI extension file must contain at least one `tab` element.

A `tab` element specifies either a new custom ribbon tab or a built-in ribbon tab that you want to customize.

Each `tab` element must contain at least one `group` element.



Attributes Each `tab` element has the following attributes:

Attribute	Description
key (Mandatory)	The unique key of the ribbon tab. ▪ If the user interface does not yet contain a ribbon tab with the specified `key` attribute, a new custom ribbon tab is created.

Attribute	Description
	▪ If the user interface already contains a ribbon tab with the specified `key` attribute, this ribbon tab is referenced. All the other `tab` attributes are ignored. To get the keys of the built-in ribbon tabs, refer to Built-in tabs and their keys. The attribute must be specified as a string value.
`insertBefore` (Optional)	The unique key of the ribbon tab that will be located to the right of the new custom ribbon tab. To get the keys of the built-in ribbon tabs, refer to Built-in tabs and their keys. If you do not specify this attribute, the new custom ribbon tab will be inserted to the right of the rightmost ribbon tab. The attribute must be specified as a string value.
`caption` (Optional)	The title of the ribbon tab. This attribute *must* be specified if you specify a new custom ribbon tab. The attribute must be specified as a string value.
`keytip` (Optional)	The keyboard tip[1)] of the ribbon tab. If you do not specify this attribute, a default keyboard tip is selected. The attribute must be specified as a string value.
`visible` (Optional)	The initial visibility of the ribbon tab. If you do not specify this attribute, the ribbon tab is visible (default = `True`). The attribute must be specified as a Boolean value.

[1)] A keyboard tip is a shortcut key active and displayed on the ribbon when the user presses the `Alt` key.

Built-in tabs and their keys The table below shows ControlDesk's built-in ribbon tabs and their keys to be used in connection with the `insertBefore` attribute.

Built-In Ribbon Tab	Key
Automation	`ControlDeskNG.TabAutomation`
Home	`ControlDeskNG.TabHome`
Layouting	`ControlDeskNG.TabLayouting`
Platforms	`ControlDeskNG.TabPlatforms`
Signal Editor	`ControlDeskNG.TabSignalEditor`
View	`ControlDeskNG.TabView`
XIL API EESPort	`ControlDeskNG.TabXILAPIEESPort`

Example The listing below shows an example:

```
<tab key="ExampleCustomUI.TabProject" caption="Project" keytip="J" insertBefore="ControlDeskNG.TabView">
```

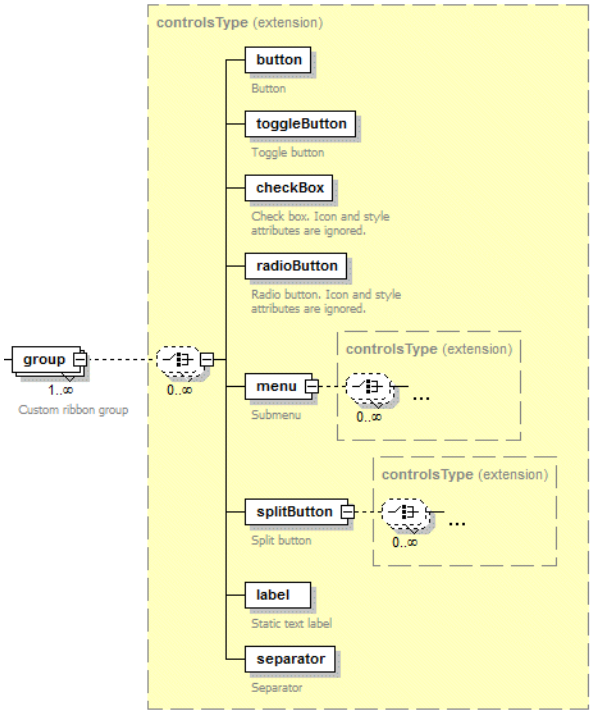
group element

The custom UI extension file must contain at least one `tab` element with at least one `group` element.

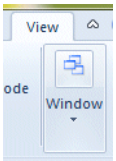
A `group` element defines a new custom ribbon group to be added to the related ribbon tab.

A custom ribbon group can contain any sequence of:

- Ribbon controls
- Labels
- Separators



Attributes Each `group` element has the following attributes:

Attribute	Description
`key` (Mandatory)	The unique key of the custom ribbon group. The attribute must be specified as a string value.
`insertBefore` (Optional)	The unique key of the ribbon group that will be located to the right of the new ribbon group. To get the keys of the built-in ribbon groups, refer to Built-in groups and their keys. If you do not specify this attribute, the new custom ribbon group will be inserted to the right of the rightmost ribbon group. The attribute must be specified as a string value.
`caption` (Mandatory)	The title of the ribbon group. The attribute must be specified as a string value.
`visible` (Optional)	The initial visibility of the ribbon tab. If you do not specify this attribute, the ribbon tab is visible (default = `True`). The attribute must be specified as a Boolean value.
`icon` (Optional)	The path to the icon file[1] relative to the path of the custom UI extension file. The icon file is displayed if there is not enough space for the ribbon group and it needs to be compressed. The illustration below shows the compressed Window ribbon group and the related icon file as an example:  The attribute must be specified as a string value.

[1] PNG or BMP file. Icon files should be transparent and have a size of 16x16 or 32x32 pixels.

Built-in groups and their keys The table below shows ControlDesk's built-in ribbon groups and their keys to be used in connection with the `insertBefore` attribute.

Built-In Ribbon Tabs and Groups	Key
Automation	`ControlDeskNG.TabAutomation`
Edit Script	`ControlDeskNG.GroupEditScript`
Interpreter	`ControlDeskNG.GroupInterpreter`
Python Scripts	`ControlDeskNG.GroupPythonScripts`
User Functions	`ControlDeskNG.GroupUserFunctions`
Home	`ControlDeskNG.TabHome`
Bookmark	`ControlDeskNG.GroupBookmark`
Calibration	`ControlDeskNG.GroupCalibration`
Clipboard	`ControlDeskNG.GroupClipboard`
Recording	`ControlDeskNG.GroupRecording`
Status Control	`ControlDeskNG.GroupStatusControl`
Layouting	`ControlDeskNG.TabLayouting`
Arrange	`ControlDeskNG.GroupArrange`
Connections	`ControlDeskNG.GroupConnections`
Instruments	`ControlDeskNG.GroupInstruments`
Layer	`ControlDeskNG.GroupLayer`
Layouts	`ControlDeskNG.GroupLayouts`
Platforms	`ControlDeskNG.TabPlatforms`
Experiment	`ControlDeskNG.GroupExperiment`
Platform Management	`ControlDeskNG.GroupPlatformManagement`
Views	`ControlDeskNG.GroupViews`
Signal Editor	`ControlDeskNG.TabSignalEditor`
Signal Generators	`ControlDeskNG.GroupSignalGenerators`
View	`ControlDeskNG.TabView`
Controlbar	`ControlDeskNG.GroupControlbar`
Show	`ControlDeskNG.GroupShow`
View Set	`ControlDeskNG.GroupViewSet`
Window	`ControlDeskNG.GroupWindow`
XIL API EESPort	`ControlDeskNG.TabXILAPIEESPort`
Error Configurations	`ControlDeskNG.GroupErrorConfigurations`
EESPorts	`ControlDeskNG.GroupEESPorts`

Example The listing below shows an example:

```
<group key="ExampleCustomUI.GroupProject" caption="Project + Experiment">
```

Ribbon control elements

A `group` element can contain any sequence of the following ribbon controls:

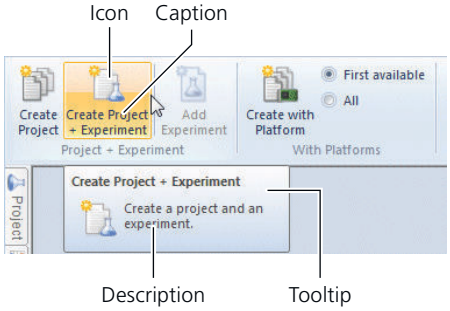
- `button` element
- `toggleButton` element

- `checkBox` element
- `radioButton` element
- `menu` element

  A `menu` element can contain any ribbon control elements.
- `splitButton` element

  A `splitButton` element has a submenu providing ribbon control elements.

The illustration below shows an example of a `button` element:



Attributes Each ribbon control element has the following attributes:

Attribute	Description
`key` (Mandatory)	The unique key of the ribbon control. The attribute must be specified as a string value.
`caption` (Mandatory)	The title of the ribbon control. The attribute must be specified as a string value.
`tooltip` (Optional)	The title of the ribbon control's tooltip. The attribute must be specified as a string value.
`description` (Optional)	The description of the ribbon control's tooltip. The attribute must be specified as a string value.
`icon` (Optional)	The path to the icon file[1)] relative to the path of the custom UI extension file. The attribute must be specified as a string value.
`keytip` (Optional)	The keyboard tip[2)] of the ribbon control. If you do not specify this attribute, a default keyboard tip is selected. The attribute must be specified as a string value.
`style` (Optional)	The display style of the ribbon control. The following styles are available: ▪ `automatic`: The style is selected automatically. ▪ `text`: Only the title of the ribbon control is displayed. ▪ `icon`: Only the icon of the ribbon control is displayed. ▪ `icontext`: The title is displayed on the right side of a 16x16 icon. ▪ `bigicontext`: The title is displayed below a 32x32 icon. If you do not specify this attribute, the ribbon control style is `automatic`. The attribute must be specified as an enumeration value.
`visible` (Optional)	The initial visibility of the ribbon control. If you do not specify this attribute, the ribbon tab is visible (default = `True`). The attribute must be specified as a Boolean value.
`enabled` (Optional)	The initial state of the ribbon control (enabled or disabled). If you do not specify this attribute, the ribbon tab is enabled (default = `True`). The attribute must be specified as a Boolean value.

Attribute	Description
`checked` (Optional)	This attribute is available only for the following ribbon control elements: ▪ `toggleButton` element ▪ `checkBox` element ▪ `radioButton` element The initial on/off state of the ribbon control element. **Note** When you click a ribbon control element, its on/off state does *not* change. You can change the state via tool automation only. The listing below shows an example: `Application.MainWindow.CustomCommands["ExampleCustomUI.Button"].IsChecked = True` If you do not specify this attribute, the ribbon control is off (default = `False`). The attribute must be specified as a Boolean value.

[1] PNG or BMP file. Icon files should be transparent and have a size of 16x16 or 32x32 pixels.
[2] A keyboard tip is a shortcut key active and displayed on the ribbon when the user presses the `Alt` key.

Example The listing below shows an example:

```
<button key="ExampleCustomUI.ButtonProjectExperiment" caption="Create Project + Experiment"
                tooltip="Create Project + Experiment" description="Create a project and an experiment."
                keytip="E" icon="ProjectExperiment.png" style="bigicontext"/>
```

Label element

A `group` element can contain any sequence of `label` elements.

A label is static text displayed in a ribbon group.

Attributes Each `label` element has the following attribute:

Attribute	Description
`caption` (Mandatory)	The text of the label in the ribbon. The attribute must be specified as a string value.

Separator element

A `group` element can contain any sequence of `separator` elements.

A separator is used to separate ribbon controls in a ribbon group.

Attributes `Separator` elements have no attributes.

Related topics

Basics

Basics on Customizing Ribbons via Extension Scripts..... 16

Examples

Example of Customizing Ribbons via Extension Scripts..... 18

Extending the Context Menu of Elements

Where to go from here

Information in this section

Extending the Context Menu of Elements

Introduction

ControlDesk's automation interface provides events that let you extend the context menu of certain elements such as experiments, variable descriptions, or layouts.

- You can extend the context menu of file-type elements in the **Project** controlbar that are represented by files such as:
  - Data sets
  - Experiments
  - Layouts and instruments
  - Measurement data files
  - Projects
  - Python scripts
  - Signal description sets
  - Variable descriptions
- You can extend the context menu of the following elements:
  - Nodes in the **Bus Navigator** controlbar
  - Nodes in the **Variables** controlbar

Note

- You cannot extend the context menu of platforms and devices.
- When you work with array instruments such as the Variable Array, you can extend the context menu of the entire array instrument only. You cannot extend the context menu of the individual array instrument elements.

Structuring context menu extensions hierarchically You can also add submenus to structure context menu extensions *hierarchically*.

Events for extending the context menu of elements

ControlDesk's automation interface provides the following events that allow you to extend the context menu of elements:

- The `ContextMenuOpening` event lets you specify the entries in the context menu.
- The `CommandExecuted` event lets you specify the command(s) to be carried out.

Example

The listing below shows how to extend the context menu of elements in the Project controlbar:

- Via the `ContextMenuOpening` event:
  - The Print Selection command is added to the context menu of all the elements in the Project controlbar.
  - The Custom Instrument Commands submenu is added to the context menu of instruments. The submenu structures the context menu extension hierarchically. The submenu contains the Bring To Front command.
- Via the `CommandExecuted` event, the commands to be carried out are specified:
  - When you invoke the Print Selection command, the name of the selected element or file is printed to the Interpreter controlbar.
  - When you invoke the Bring To Front command, the instrument selected in the layout is brought to the front.

```
import win32com.client
import dspace.com
def On_Application_ContextMenuOpening(Menu, Selection):
   """
   Syntax : On_Application_ContextMenuOpening
   Purpose:
   Parameters: Menu, Selection
   """
   print "On_Application_ContextMenuOpening"
   Menu.AddSeparator()
   Cmd = Menu.AddCommand()
   Cmd.Key = "PrintSelection"
   Cmd.Caption = "Print Selection"
   if len(Selection) > 0:
      ComIdentity = dspace.com.GetComIdentity(Selection[0])
      if ComIdentity.startswith("IVi") and ComIdentity.endswith("Instrument"):
         Menu.AddSeparator()
         SubMenu = Menu.AddSubMenu("Custom Instrument Commands")
         Cmd = SubMenu.AddCommand()
         Cmd.Key = "BringToFront"
         Cmd.Caption = "Bring To Front"
```

```
def On_Application_CommandExecuted(Command, Selection):
   """
   Syntax : On_Application_CommandExecuted
   Purpose: Parameters: Command, Selection
   """
   print "On_Application_CommandExecuted"
   print Command.Key + " fired!"
   if Command.Key == "PrintSelection":
      print "Selection:"
      for Obj in Selection:
         ComType = dspace.com.GetComIdentity(Obj)
         if ComType == "IXaFile":
            print "IXaFile: " + Obj.FullPath
         elif ComType == "IXaExperiment" or\
              ComType == "IXaActiveExperiment" or\
              ComType == "IXaActiveProject" or\
              ComType == "IXaLayoutDocument" or\
              (ComType.startswith("IVi") and ComType.endswith("Instrument")):
            print "%s: %s" % (ComType, Obj.Name)
         else:
            print ComType
   elif Command.Key == "BringToFront":
      for Obj in Selection:
         ComType = dspace.com.GetComIdentity(Obj)
         if ComType.startswith("IVi") and ComType.endswith("Instrument"):
            Obj.MoveToTop()
```

Related topics

Basics

Basics on Using ControlDesk Events (ControlDesk Automation 🕮)

HowTos

How to Assign Python Code to an Event (ControlDesk Automation 🕮)

Example of Extending the Context Menu of a Layout via an Extension Script

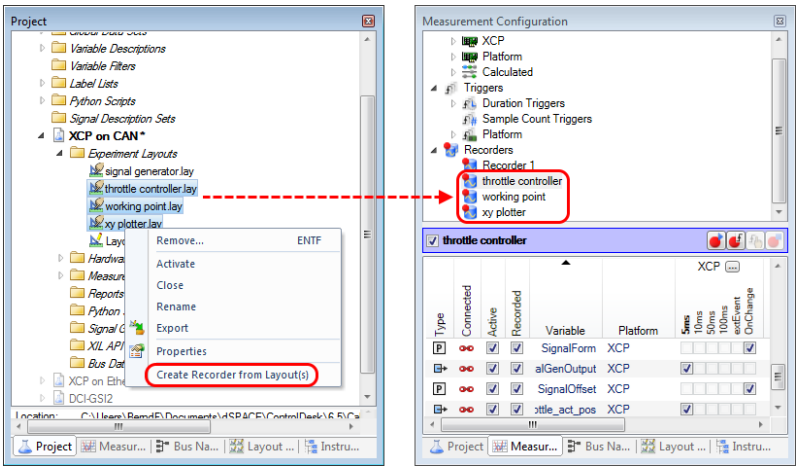
Example overview

What you will learn The example shows you:

- How to add a new command to the context menu of each layout in a project via the `ContextMenuOpening` event.
  - For layout files in the Project controlbar, the new context menu command is called Create Recorder from Layout(s).
  - For layouts in the working area, the new context menu command is called Create Recorder from Layout.
- How to specify the new context menu command via the `CommandExecuted` event.
  - New recorders are created in the measurement configuration named after the selected layouts.

- Signals are added to the new recorders. Each recorder gets the signals that are connected to the instruments of the associated layout.

Refer to the following illustration (for an animated graphic, refer to dSPACE Help):



Example based on CustomUI demo The example described below is based on the *ContextMenuExample* extension script of the CustomUI tool automation demo.

To extend the ControlDesk ribbon by using this demo, perform the following steps:

1. From the `.\Demos\ToolAutomation\Python\CustomUI` folder of your ControlDesk installation, copy the `ContextMenuExample.extscript` file and the entire `ContextMenuExample` folder to the following folder:

   `%USERPROFILE%\Documents\dSPACE\ControlDesk\7.4\ExtensionScripts`
2. Restart ControlDesk.

Tip

As an alternative, you can use an installation script that installs both examples of the CustomUI demo (*RibbonTabExample* and *ContextMenuExample*) in one step. Refer to the `info.txt` file in the `CustomUI` demo folder.

The following listings show excerpts from the `ContextMenuExample.py` file.

Adapting the script to your requirements

The following listing shows the flags you can change in the script to adapt it to your requirements.

```
#----------------------------------------------------------------
# Define global flags used in this demo. This flags could be changed.
#----------------------------------------------------------------
# Defining whether the layouts which are closed should be closed again after processing.
CloseLayouts = True
# Defining whether only measurement variable should be added to a recorder.
OnlyMeasurementsVariables = False
# Defining whether an existing recorder will be removed and a new one will be created.
OverwriteExistingRecorder = True
# Defining whether all log informations should be written to log.
Verbose = False
```

Adding a new command to the context menu

The following listing shows you how to add a new command to the context menu of a layout in the working area or a layout file in the Project controlbar.

- If you open a context menu of layout in the working area, the command Create Recorder for Layout is added.
- If you open a context menu of a layout file in the Project controlbar, the command Create Recorder from Layout(s) is added because multiple selection is possible.

```
class ApplicationEvents(object):
(...)
   def OnContextMenuOpening(self, menu, selection):
      """This method will be called, if a context menu has been opened by the user.
      """
      menu = Dispatch(menu)
      selectionCount = len(selection)
      layoutCount = 0
      layoutFilesCount = 0
      # Analyse the content of the selection.
      for item in selection:
          (...)
          if dspace.com.GetComIdentity(item) == "IXaLayoutDocument":
             layoutCount = layoutCount + 1
          elif dspace.com.GetComIdentity(item) == "IXaFile" and item.Type == Enums.FileType.Layout:
             layoutFilesCount = layoutFilesCount + 1
      # Choose which context menu must be extended.
      if layoutCount > 0 and layoutCount == selectionCount:
          # This is a context menu inside a layout.
          menu.AddSeparator()
          CommandCreateRecorderFromLayout = menu.AddCommand()
          CommandCreateRecorderFromLayout.Key = COMMANDKEYCREATERECORDERFROMLAYOUT
          CommandCreateRecorderFromLayout.Caption = "Create Recorder from Layout"
          (...)
      elif layoutFilesCount > 0 and layoutFilesCount == selectionCount:
          # This is a context menu on experiment layout documents.
          menu.AddSeparator()
          CommandCreateRecorderFromLayout = menu.AddCommand()
          CommandCreateRecorderFromLayout.Key = COMMANDKEYCREATERECORDERFROMLAYOUTS
          CommandCreateRecorderFromLayout.Caption = "Create Recorder from Layout(s)"
```

Creating new recorders when the command is executed

The following listing shows you how to create new recorders for one ore more selected layouts or layout files when the new context menu command is executed.

```
COMMANDKEYCREATERECORDERFROMLAYOUT = "CreateRecorderFromLayout.CommandCreateRecorderFromLayout"
COMMANDKEYCREATERECORDERFROMLAYOUTS = "CreateRecorderFromLayout.CommandCreateRecorderFromLayouts"
(...)
class ApplicationEvents(object):
   (...)
   def OnCommandExecuted(self, command, selection):
      """This method will be called, if a command has been executed by the user."""
      (...)
      if command.Key == COMMANDKEYCREATERECORDERFROMLAYOUT:
         CreateRecorderFromLayouts(selection)
      elif command.Key == COMMANDKEYCREATERECORDERFROMLAYOUTS:
         CreateRecorderFromLayoutFiles(selection)
      (...)
def CreateRecorderFromLayout(layout):
   """This function creates a recorder for the given layout document."""
   global CurrentRecorder
   recorderName = layout.Name
   (...)
   # Add the recorder.
   (...)
   CurrentRecorder = Application.MeasurementDataManagement.Recorders.Add(recorderName)
   (...)
def CreateRecorderFromLayouts(layouts):
   # Iterate through the layouts.
   (...)
   for layout in layouts:
       (...)
       CreateRecorderFromLayout(layout)
       (...)
def CreateRecorderFromLayoutFiles(layoutFiles):
   """This function creates recorders for all given layout files."""
   (...)
   for layoutFile in layoutFiles:
      (...)
      # Get/Open the layout document and process the layout.
      layout = layoutFile.Open()
      (...)
      # Create the recorder from the layout
      CreateRecorderFromLayout(layout)
```

Adding signals to the new recorders

The command iterates over all instruments and connected variables of the selected layouts to add all relevant signals to the related recorders.

```
import LayoutManagementLib
COMMANDKEYCREATERECORDERFROMLAYOUT = "CreateRecorderFromLayout.CommandCreateRecorderFromLayout"
COMMANDKEYCREATERECORDERFROMLAYOUTS = "CreateRecorderFromLayout.CommandCreateRecorderFromLayouts"
(...)
class ApplicationEvents(object):
(...)
```

```
def CreateRecorderFromLayout(layout):
   """This function creates a recorder for the given layout document."""
   (...)
   # Iterate over the instruments of the layout.
   LayoutManagementLib.IterateConnectedVariables(layout, OnConnectedVariable, WriteLogEntry, WriteErrorEntry)
   (...)
def OnConnectedVariable(instrument, subObject, variableProperty, variable):
   """Callback method that gets called for each variable connected to an instrument on a layout."""
   (...)
   if addSignal:
   # Add the signal to the signal collection and insert the returned signal object to the recorder.
   # If a signal is already in the signal collection this object will be returned by the Add method.
   signals = Application.MeasurementDataManagement.MeasurementConfiguration.Signals
   CurrentRecorder.Signals.Insert(signals.Add(path))
```

EXTSCRIPT XML configuration file example

The listing below shows the XML configuration file that references the extension script and the `ContextMenuExample.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<ExtensionScripts xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="ExtensionScripts.xsd">
   <PythonScript RelativePath="ContextMenuExample\ContextMenuExample.py" />
</ExtensionScripts>
```

Related topics

Basics

Executing Extension Scripts When ControlDesk Starts Up.....12
Tool Automation Demos (ControlDesk Introduction and Overview 🕮)

References

Schema of the Custom UI Extension File.....21

Adding User Functions to ControlDesk

Where to go from here

Information in this section

Basics on User Functions

Introduction

You can add external applications or functions to ControlDesk.

User functions

ControlDesk allows you to embed external applications or additional functions as *user functions*. A user function is available in the **Automation** ribbon as a button in the **User Functions** ribbon group.

For example, you can add the CalDemo ECU to ControlDesk as a user function and assign a drive image as the user function button.

To configure a user function, you have to specify an executable file or a Python file. Depending on the file type, ControlDesk performs the following actions when you execute the user function:

Executable files When you specify an executable file, ControlDesk starts the corresponding application in a separate process. Files with the EXE, COM, or BAT extension, for example, are executable files.

Python scripts When you specify a PY or PYC file, ControlDesk runs it in the Interactive Interpreter (see Interpreter (ControlDesk Automation 🕮)).

Note

User functions based on Python do not have access to ControlDesk's automation interface. To add functions with full access to ControlDesk's automation interface to the user interface, you can use *extension scripts*. Refer to Executing Extension Scripts When ControlDesk Starts Up on page 12.

Related topics

HowTos

How to Add External Programs or Scripts as User Functions to ControlDesk..... 35

References

Customize (User Functions) (ControlDesk Automation 🕮)
Interpreter (ControlDesk Automation 🕮)

How to Add External Programs or Scripts as User Functions to ControlDesk

Objective

ControlDesk lets you start executable files and Python scripts from within ControlDesk.

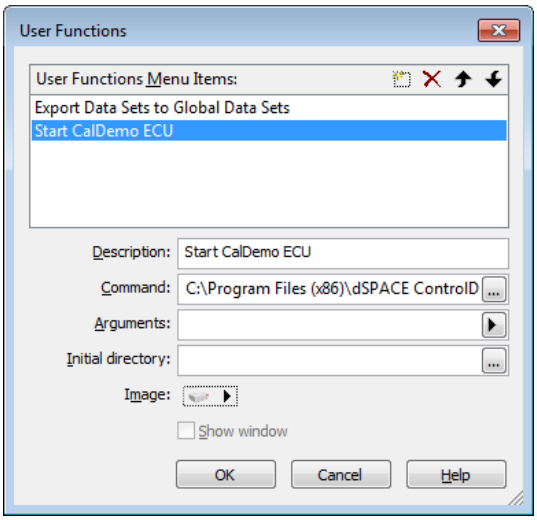
Method

To add external programs or scripts as user functions to ControlDesk

1 On the Automation ribbon, click User Functions – Customize.
The User Functions dialog opens.

2 In the dialog, click  to add a new user function.

3 Enter a description and select a Python script or an executable file and an image for the new user function.



4 Configure further settings, such as additional arguments or an initial directory.

Tip

To apply an accelerator (underlined character in the menu), prefix the character in the user function name with an ampersand (**&**), for example **`S&tart CalDemo ECU`**. Press **`Alt`** to visualize accelerators in the menu.

5 Click OK to confirm your settings and close the dialog.

Result

You have added an external program or a script as user function. A new button with the specified icon appears in the Automation – User Functions ribbon group.

Related topics

Basics

Basics on User Functions 34

References

Customize (User Functions) (ControlDesk Automation 🕮)

Customizing Instrument Handling

Where to go from here

Information in this section

Adding a Python Script to an Instrument or Layout

Where to go from here

Information in this section

Adding a Python Script to an Instrument or Layout

Introduction

You can add a Python script to an instrument or layout. This lets you extend the functionality of the instrument or layout via automation.

Script assignment

You can add one Python script to each instrument or layout.

In the script, you can assign Python code to the events of the selected instrument or layout.

Script execution

Instrument scripts and layout scripts are executed automatically each time the instrument or layout is initialized. An instrument is initialized, for example, when the layout in which it is contained is opened.

Instrument and layout scripts are executed in a common Python interpreter that is independent of the ControlDesk Internal Interpreter.

Note

The common interpreter ignores Python search paths specified on the Interpreter page. As an alternative, specify additional search paths directly in the Python installation to make them accessible to the common interpreter.

Printing script outputs

Printing script outputs to the Interpreter controlbar To print instrument or layout script outputs to the **Interpreter** controlbar, you have to use the automation of the Internal Interpreter.

The following listing shows an example:

```
Application.Interpreter.InteractiveWindow.Exec("print('Hello World.')")
```

Refer to `Interpreter / IPiInterpreter <<Interface>>` (ControlDesk Automation 🕮).

Printing script outputs to the Message Viewer To print instrument or layout script outputs to the **Messages controlbar**, you have to use its automation interface.

The following listing shows an example:

```
Application.Log.WriteInformation("Hello World.")
```

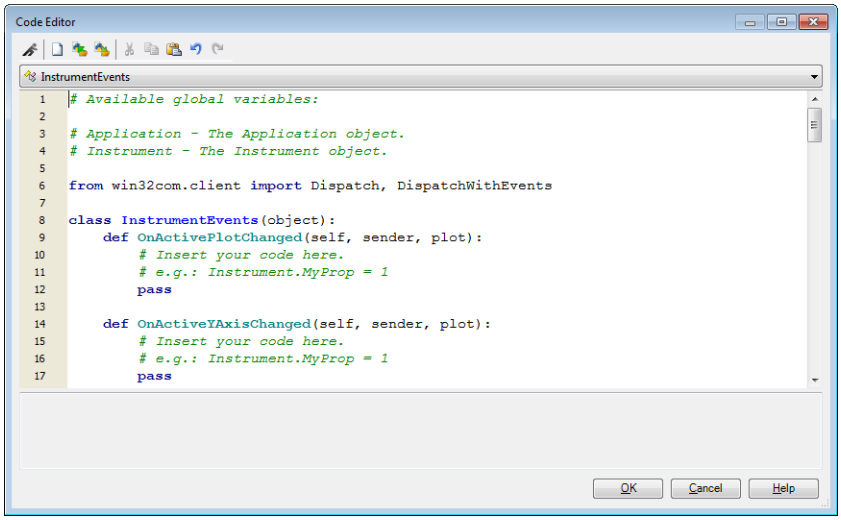
Refer to `Log / ILoLog <<Interface>>` (ControlDesk Automation 🕮).

Script storage

The script is stored in the same location as the instrument or layout. This allows you, for example, to create custom instruments with the extended functionality.

Editing Python scripts with the Code Editor

You can edit a Python script for an instrument or layout in ControlDesk's Code Editor. The Code Editor offers a template containing the events of the selected instrument or layout.



For details on the commands and buttons, refer to Code Editor Dialog (ControlDesk Instrument Handling 🕮).

Example

For an example, refer to Example of Adding a Python Script to an Instrument on page 39.

Related topics

Examples

Example of Adding a Python Script to an Instrument..... 39

References

Instrument Script (ControlDesk Instrument Handling 🕮)
Layout Properties (ControlDesk Layouting 🕮)

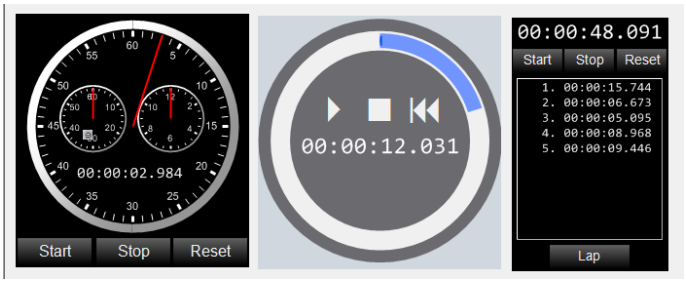
Example of Adding a Python Script to an Instrument

Introduction

The stopwatches in the Instrument Selector are examples for extending instrument functionality via instrument scripts.

Basics on ControlDesk's stopwatches

ControlDesk's Instrument Selector provides various stopwatches, as shown in the following illustration (for an animated graphic, refer to dSPACE Help).

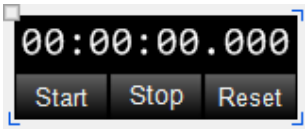


The functionality of the stopwatches, such as starting, stopping, or displaying lap times, is added to them via instrument scripts.

Each stopwatch consists of an instrument group that contains different instruments. Each single instrument can contain one instrument script. The instrument scripts of the single instruments interact and create the functionality of the stopwatch.

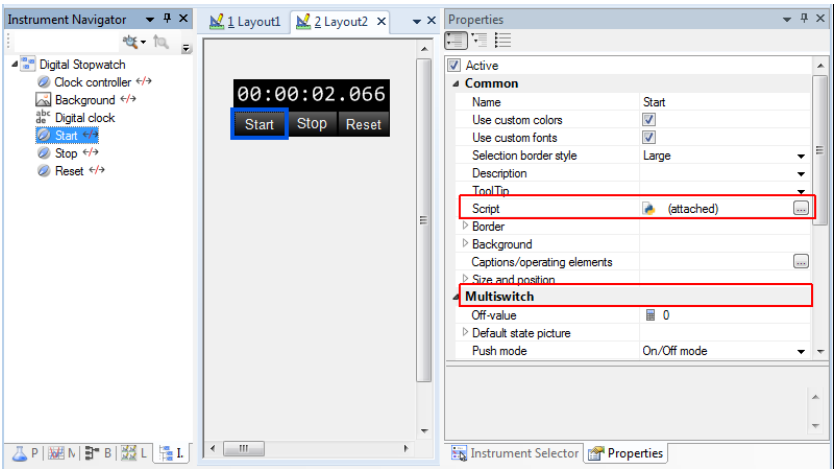
Example: Digital Stopwatch

The following example shows how instrument scripts are used for the Digital Stopwatch.



Description of the Digital Stopwatch When you place a Digital Stopwatch on a layout, the Instrument Navigator controlbar shows the single instruments of the stopwatch. To identify a single instrument of the stopwatch on the layout, you can select a Selection border style in the Properties controlbar.

The following illustration shows the stopwatch's Start button with a selection border as an example. In the Properties controlbar, you can identify this button as a Multiswitch instrument extended by an attached instrument script.

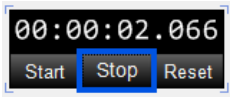


Description of the single instruments The following table explains the single instruments of the Digital Stopwatch.

Instrument	Instrument Description	Script Description

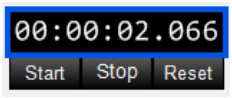
Clock controller (Multiswitch)
 
	No display purposes. The central script of the stopwatch is attached to this instrument.	To check the `clock_controller.Value`, and to call the functions to start, stop, or reset the calculation of the time to be displayed.
		Script excerpt:
		`if value == 0:`
		`  self.stopwatch.start()`
		`elif value == 1:`
		`  self.stopwatch.stop()`
		`elif value == 2:`
		`  self.stopwatch.reset()`

Start (Multiswitch)
 
	To start the stopwatch via the attached instrument script.	To check whether the instrument's active state is changed, and to set the `clock_controller.Value` to a specific value.
		Script excerpt:
		`def OnActiveStateChanged(self, sender, oldState):`
		`  if self.on_state is not None and self.on_state.IsActive:`
		`  self.clock_controller.Value = 0.0`

Stop (Multiswitch)
 
	To stop the stopwatch via the attached instrument script.	To check whether the instrument's active state is changed, and to set the `clock_controller.Value` to a specific value.
		Script excerpt:
		`def OnActiveStateChanged(self, sender, oldState):`
		`  if self.on_state is not None and self.on_state.IsActive:`
		`  self.clock_controller.Value = 1.0`

Reset (Multiswitch)
 
	To reset the stopwatch via the attached instrument script.	To check whether the instrument's active state is changed, and to set the `clock_controller.Value` to a specific value.
		Script excerpt:
		`def OnActiveStateChanged(self, sender, oldState):`
		`  if self.on_state is not None and self.on_state.IsActive:`
		`  self.clock_controller.Value = 2.0`

Background (Frame) 	Display purposes.	- (no script attached)

Instrument	Instrument Description	Script Description
Digital clock (Static text) 	To display the current time, which is calculated by the script added to the Clock controller.	- (no script attached)

Complete script examples

To read the complete scripts attached to a stopwatch, place the stopwatch on a layout, click a single instrument of the stopwatch in the Instrument Navigator and open the attached script via the Properties controlbar.

Related topics

Basics

Adding a Python Script to an Instrument or Layout 37

References

Code Editor Dialog (ControlDesk Instrument Handling 🕮)
Stopwatches Library (ControlDesk Instrument Handling 🕮)

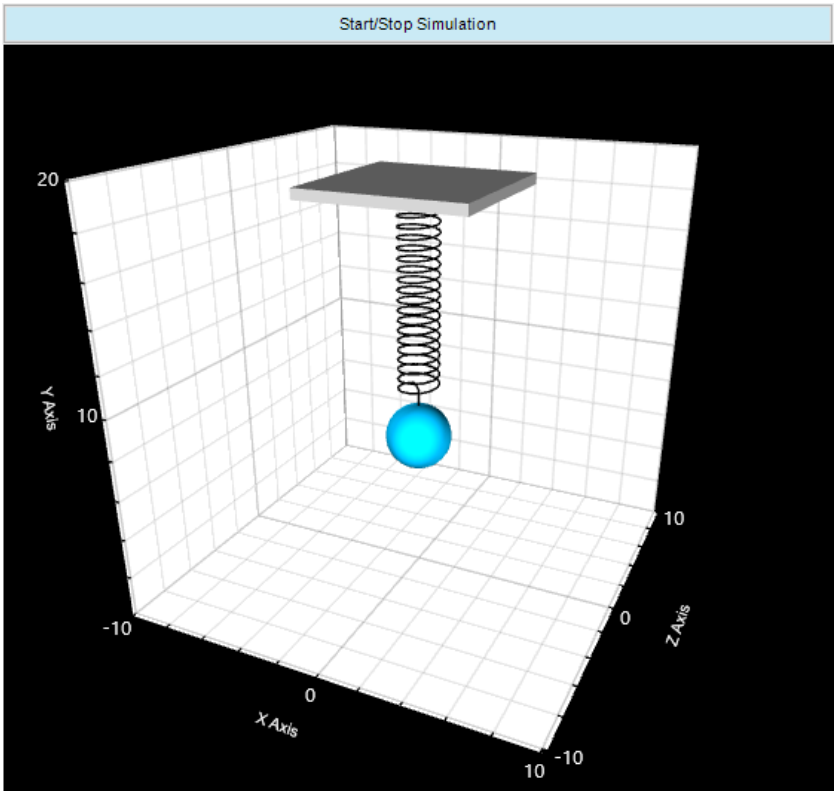
Example of Using an Instrument Script with the 3-D Viewer

Introduction

The following example is based on the Instrumentation Demo. The example visualizes the vertical movement of a mass connected to a spring.

The goal of this example is to show how to change item properties using an instrument script. The movement of items in this example is not related to the value change of connected variables, but is calculated by the instrument script that is added to a Push Button.

The following illustration shows the result of the demo (for an animated graphic, refer to dSPACE Help).



Instruments

The example uses the following instruments. Both instruments are combined in an instrument group.

Push Button The Push Button has a button labeled **Start/Stop Simulation** to start or stop the simulation.

An instrument script is added to the Push Button. When you click the button, the instrument script is executed.

3-D Viewer The 3-D Viewer visualizes the anchor, the spring and the mass.

Object	Item Type	Description
Anchor	Cuboid	Because the anchor does not move, the item properties are defined once by the instrument script and are not altered afterwards.
Spring	Point Line	The spring is hanging from the anchor. For every simulation step, the new positions of the points are calculated by the instrument script.
Mass	Point Line	The mass is represented by a point line with only one point. For every simulation step, the new position of the point is calculated by the instrument script.

Starting the simulation

When you click the Start/Stop Simulation button, the `ButtonClicked` event occurs and the `OnButtonClicked` function is executed.

For more information on the event, refer to PushButtonInstrumentsEvents / IViPushButtonInstrumentsEvents <<EventInterface>> (ControlDesk Automation 🕮).

The following script is executed as a part of the `OnButtonClicked` function.

```
if self.simulation_timer is None and self.demo_instrument is not None:
    self.simulation_init()
    self.simulation_timer = SimulationTimer(self.simulation_step, self.update_interval)
    self.simulation_timer.start()
```

If no simulation is running, a `SimulationTimer` object is created with the `simulation_step` function and the `update_interval` as parameters.

According to the following script, a thread is started to execute the `run` function.

```
class SimulationTimer(Thread):
    (…)
    def __init__(self, func, interval):
        """Initialize the timer.
        func: the function called by the timer
        interval: the interval in seconds between function calls
        """
        Thread.__init__(self)
        self.func = func
        self.interval = interval
        self.finished = Event()
        self.daemon = True
    def cancel(self):
        """Cancel the timer.
        """
        self.finished.set()
    def run(self):
        """Run the timer.
        """
        pythoncom.CoInitialize()

        while True:
            self.finished.wait(self.interval)
            if self.finished.is_set():
                break

            self.func()

        pythoncom.CoUninitialize()
```

Refer to Multithreaded Scripting (ControlDesk Automation 🕮).

As a result, the `simulation_step` function is called repeatedly. The parameter `update_interval` specifies the time period between the steps.

Single simulation step

As part of a single simulation step, the `simulation_step` function is called to calculate the vertical displacement of the mass relative to the default position. The `simulation_step` function calls the `set_values` function to calculate the

positions of the spring and the mass depending on the displacement `length_delta`.

simulation_step The `simulation_step` function calculates the displacement of the mass. To calculate the positions of the spring and the mass, the `set_values` function is called. Depending on the number of steps, the `damping_increment` is calculated, or the simulation is canceled if the end of the simulation is reached.

```
def simulation_step(self):
    (…)
    #Calculate the displacement of the mass relative to the default position
    length_delta = math.sin(self.step_angle) * self.spring_length_max_delta * self.damping_factor

    #Calculate the positions of the spring and the mass
    self.set_values(length_delta)

    #Calculate the damping for the next simulation step or cancel the simulation
    self.step_angle += self.step_angle_increment

    if self.step <= self.step_count:
        self.damping_factor += self.damping_increment
        self.step += 1

    elif self.step <= 2 * self.step_count:
        self.damping_factor -= self.damping_increment
        self.step += 1

    else:
        self.simulation_timer.cancel()
        self.simulation_timer = None
```

set_values The `set_values` function calculates the positions of the points for the spring and the mass for a given `length_delta` displacement. The position values of the points are added to a set of arrays. When all positions are calculated, the arrays are used to add the points to the point lines.

```
@locking
    def set_values(self, length_delta):
        (…)
        demo = self.demo_instrument
        (…)

        spring = demo.Items.PointLines[0]
        mass = demo.Items.PointLines[1]
        # Remove the points of the spring and the mass
        spring.Points.Clear()
        mass.Points.Clear()
        # Define arrays to save the positions of the spring points
        spring_x = []
        spring_y = []
        spring_z = []
        # Define various helpful variables
        spring_end_y = self.spring_length_base_position + length_delta
        point_count = math.floor((2 * math.pi * self.windings_count) / self.spring_angle_increment)

        y_value = self.spring_ypos_anchor
        y_increment = (self.spring_ypos_anchor - spring_end_y) / point_count
        spring_angle = 0
        spring_radius = 1
```

```
        # Calculate the positions of the spring points and add them to the arrays
        for i in range(point_count):
            spring_x.append(math.sin(spring_angle) * spring_radius)
            spring_y.append(y_value)
            spring_z.append(math.cos(spring_angle) * spring_radius)

            spring_angle += self.spring_angle_increment
            y_value -= y_increment
        # Add two additional points to the spring
        spring_x.append(0)
        spring_z.append(0)
        spring_y.append(spring_y[-1])

        spring_x.append(0)
        spring_z.append(0)
        spring_y.append(spring_y[-2] - 2)
        # Define arrays to save the position of the mass
        mass_x = [0]
        mass_y = [spring_y[-1] - .4]
        mass_z = [0]
        # Use the arrays to add the points to the spring and the mass
        spring.Points.AddRange(spring_x, spring_y, spring_z)
        mass.Points.AddRange(mass_x, mass_y, mass_z)
```

Avoid item flickering via locking decorator

When you start the simulation using the previously shown instrument script, the spring and the mass flicker.

While calculating the position values for the items, there is a brief period of time when all position values are cleared and both point lines are not visible. The repeated appearance and disappearance of the point lines lead to flickering. To fix this, the `locking` decorator is defined to lock the instrument until the new points are added. The `locking` decorator is applied to the set_values function. Whenever the function is called, the 3-D Viewer will lock before executing the function and unlock afterwards.

```
def locking(func):
    (…)
    def wrapper(*args, **kwargs):
        self_obj = args[0]
        instr = self_obj.demo_instrument

        try:
            instr.Lock()
            func(*args, **kwargs)
        finally:
            instr.Unlock()

    return wrapper
```

Stopping the simulation

When you click the Start/Stop Simulation button while a simulation is running, the following script is executed as a part of the `OnButtonClicked` function.

```
elif self.simulation_timer is not None:
    self.simulation_timer.cancel()
    self.simulation_timer.join()
    self.simulation_timer = None
    self.set_values(0)
```

The script cancels the simulation thread and resets the spring and the mass to their default position.

Result

To simulate a mass hanging from a spring, an instrument script is added to a Push Button. When you click a button, the instrument script is executed.

- When you click the **Start/Stop Simulation** button while no simulation is running, the script starts a thread to repeatedly calculate new positions for two point lines representing the mass and the spring. When the simulation is finished, the simulation thread is canceled.
- When you click the **Start/Stop Simulation** button while a simulation is running, the script cancels the simulation thread and resets the spring and the mass to their default position.

Related topics

Basics

Basics of Handling the 3-D Viewer (ControlDesk Instrument Handling 🕮)
Instrumentation Demo (ControlDesk Introduction and Overview 🕮)

Examples

Example of Using the 3-D Viewer (ControlDesk Instrument Handling 🕮)

References

3-D Viewer (ControlDesk Instrument Handling 🕮)

Adding Custom Properties to an Instrument

Where to go from here

Information in this section

Basics on Adding Custom Properties to an Instrument

Introduction

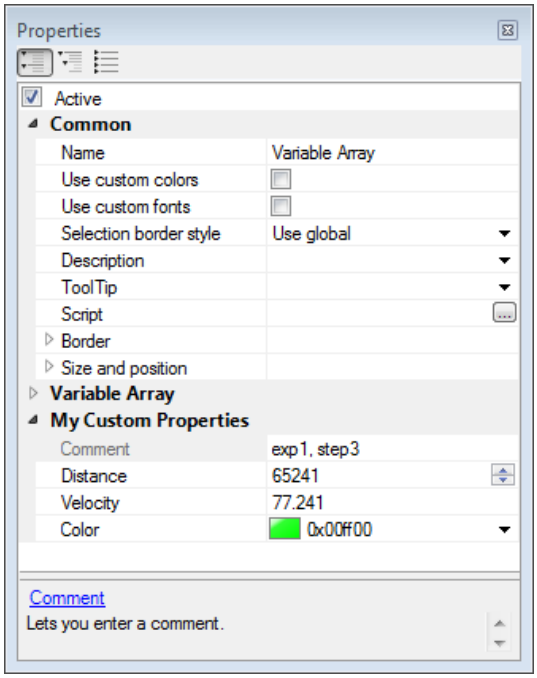
You can add various types of custom properties to an instrument via automation.

Custom properties

Adding custom properties allows you to extend the properties list of an instrument. Custom properties are user-defined properties that you add to an instrument via an automation script or via the **Interpreter**.

After custom properties are added to an instrument, they are part of the instrument. You can then let an instrument script be executed upon the occurrence of an event indicating that a custom property of an instrument has been changed.

The following illustration shows the **Properties** controlbar of a Variable Array containing a user-defined property category named **My Custom Properties**.

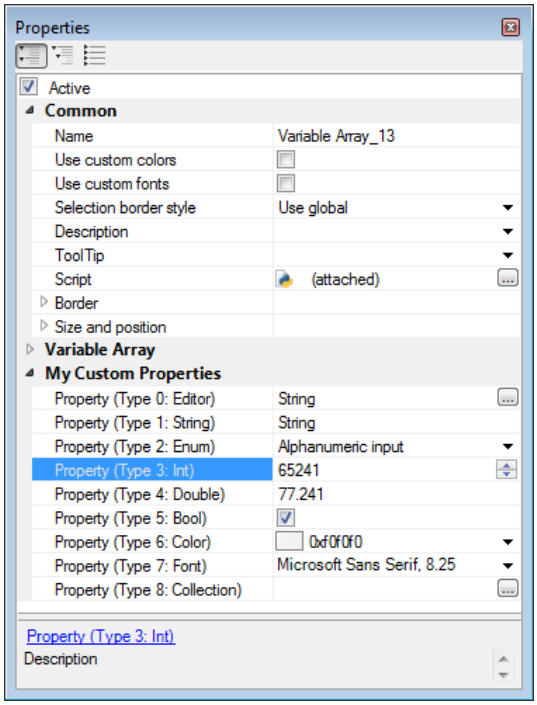


Custom property types

When you add a custom property to an instrument, you must specify the property type via an enumeration value.

Custom Property Type	Description	Value
Editor	The property offers an edit field for a string and a Browse button to open an external dialog.	0
String	The property offers an edit field for a string.	1
Enum	The property offers a list of items to select one item.	2
Int	The property offers an edit field for an integer value.	3
Double	The property offers an edit field for a double value.	4
Bool	The property offers a checkbox to enable or disable the property.	5
Color	The property offers a color field to open a color dialog.	6
Font	The property offers a font field to open a font dialog.	7
Collection	The property offers a Browse button to open a Properties dialog displaying a list of items. In the dialog, you can add and delete items and configure their properties.	8

The following illustration shows an example in which the names of the custom properties contain the type of the custom property.



Properties of custom properties

You can get or set various properties for a custom property. This lets you, for example, add the custom property to a specific category or make it read-only.

Property	Property Type	Description	Data Type
AlphaChannelSupported	Color	(Available only for a custom property of `Color` type). Gets or sets a value indicating whether the alpha channel is supported.	*Boolean*
Category	All	Gets or sets the category name. It specifies the category that the custom property is assigned to in the **Properties** controlbar.	*String*
Count	Collection	Gets the number of elements.	*Signed 32 Bit Integer*
Description	All	Gets or sets the description.	*String*
EnumValues	Enum	Gets or sets the enum values.	*System.Object[]*
Name	All	Gets or sets the name of the custom property.	*String*
ReadOnly	All	Gets or sets a value indicating whether the custom property is read-only.	*Boolean*
Tag	All	Gets or sets a custom tag.	*String*
Visible	All	Gets or sets a value indicating whether the custom property is visible.	*Boolean*
Value	All, except for collection	Gets or sets the value.	Depends on the property type

Refer to the **CustomProperty<Type>** topics for detailed information, for example, CustomPropertyColor (refer to CustomPropertyColor / IViCustomPropertyColor <<Interface>> (ControlDesk Automation 🕮)).

Adding custom properties

For information on adding custom properties to an instrument and code examples, refer to Adding Custom Properties to an Instrument on page 50.

Related topics

References

CustomProperties / IViCustomProperties <<Interface>> (ControlDesk Automation 🕮)

Adding Custom Properties to an Instrument

Introduction

You can extend the functionality of an instrument by adding custom properties to it.

Basics

For basic information on custom properties and the types they can have, refer to Basics on Adding Custom Properties to an Instrument on page 48.

Adding custom properties

You add custom properties via the `Add` method of the instrument's `CustomProperties` interface. The parameters of the `Add` method specify the type and the name of the property:

```
NewProperty = Instrument.CustomProperties.Add(6, 'Default Color')
```

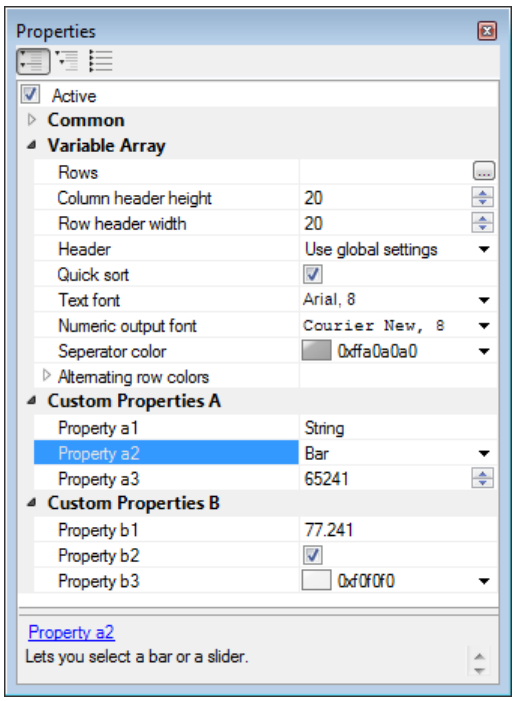
Refer to:

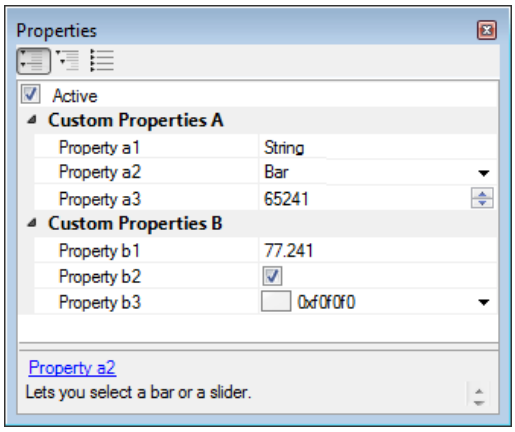
- CustomProperties / IViCustomProperties <<Interface>> (ControlDesk Automation 🕮)
- CustomPropertyTypeConst <<Enumeration>> (ControlDesk Automation 🕮)

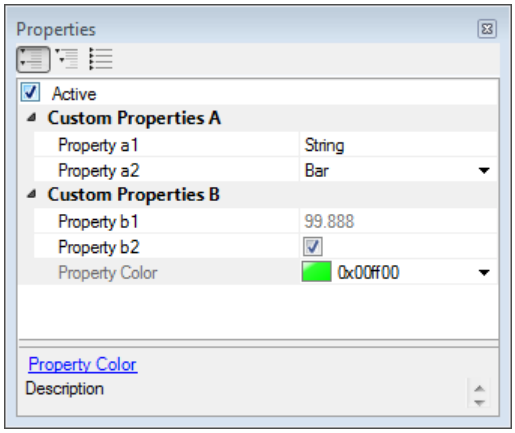
Configuring properties

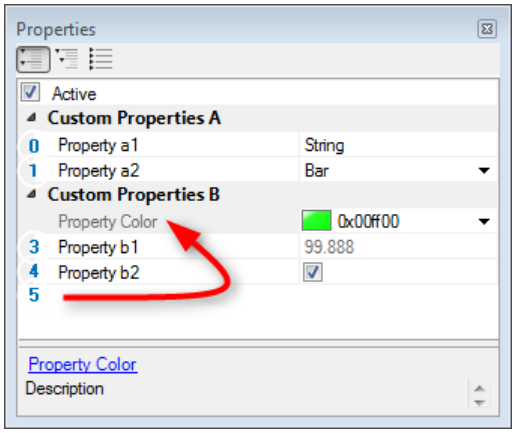
You can use various properties or methods of the `CustomProperties` interface to configure the properties list. The following table shows some examples.

Properties Controlbar	Code Snippet

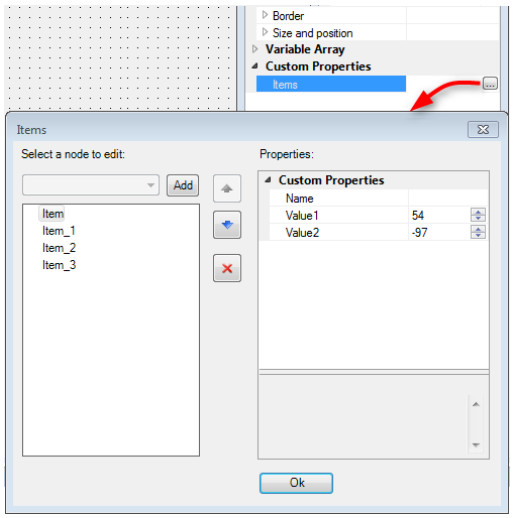
	You can assign custom properties to different categories. You can also create new categories this way.
	The description string is displayed in the embedded help region at the bottom of the Property Grid.
	`...`
	`# Property enum type`
	`p = Instrument.CustomProperties.Add(2, 'Property a2')`
	`p.Category = 'Custom Properties A'`
	`p.Description = 'Lets you select a bar or a slider.'`
	`p.EnumValues = ['Bar', 'Slider']`
	`p.Value = 'Bar'`

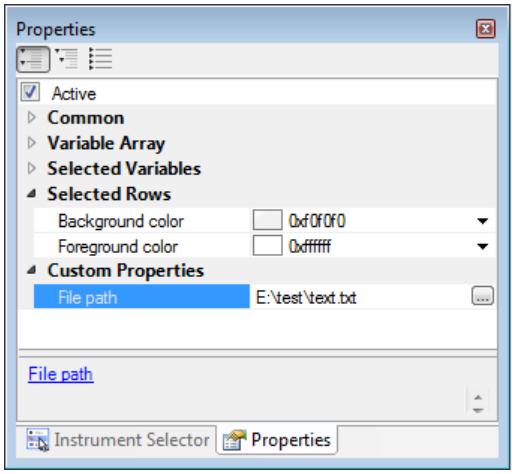
Properties Controlbar	Code Snippet
	You can hide the default (non-custom) properties. `Instrument.CustomProperties.DefaultPropertiesVisible=False`

	You can change various properties of a custom property, for example, change its value, make it read-only, or hide it.
	Keep in mind that the index of the items starts with 0.
	`Instrument = ActiveLayout.Instruments["Variable Array"]`
	`pa3 = Instrument.CustomProperties.Item(2)`
	`pb1 = Instrument.CustomProperties.Item(3)`
	`pb3 = Instrument.CustomProperties.Item(5)`
	`pa3.Visible = False`
	`pb1.Value = 99.888`
	`pb1.ReadOnly = True`
	`pb3.Name = "Property Color"`
	`pb3.Value = 0x00ff00`

	You can change the order of the properties. Keep in mind that invisible items are still part of the index. In the example, Property a3 with index number 2 is invisible. `Instrument.CustomProperties.MoveTo(5,3)`

Properties Controlbar	Code Snippet

	You can define a collection of items with the same properties. In the Properties controlbar, the user can edit the item list in a dialog.
	`Collection = instrument.CustomProperties.Add(8,'Items')`
	`# Define a template for the custom properties`
	`Collection.Template.Add(1,'Name')`
	`Collection.Template.Add(3,'Value1')`
	`Collection.Template.Add(3,'Value2')`
	`# Add items to item list`
	`Collection.Items.Add()`
	`Collection.Items.Add()`
	`Collection.Items.Add()`
	`Collection.Items.Add()`
	`# Specify the item's properties`
	`a = Collection.Items[0].Item(1)`
	`a.Value = 54`
	`b = Collection.Items[0].Item(2)`
	`b.Value = -97`

	You can use an editor type property to offer the user the possibility to specify a file path via a dialog that you have created. `p = Instrument.CustomProperties.Add(0, 'File path')`

Workflow

Follow these steps to add custom properties via an automation script or via the Interpreter:

1. Add a new property to the instrument's custom properties using the `CustomProperties.Add` method.

   The parameters of the Add method are the property type and the property name. In the following example, the property name contains the type information:

   ```
   NewProperty = Instrument.CustomProperties.Add(3, 'Property 01')
   ```

2. Specify the specific properties of the selected custom property type, for example:

   ```
   NewProperty.Category = 'My Custom Properties'
   NewPorperty.Description = 'Description'
   NewProperty.Value = 65241
   ```

3. Add a Python script to the instrument and use the `OnCustomPropertyChanged` instrument event to specify the code that is executed when the property value is changed.

Custom property in instrument script

For information on using the `OnCustomPropertyChanged` instrument event in a Python script, refer to Relating Custom Property Changes to Instrument Script Execution on page 54.

Related topics

References

CustomProperties / IViCustomProperties <<Interface>> (ControlDesk Automation 🕮)
CustomPropertyTypeConst <<Enumeration>> (ControlDesk Automation 🕮)

Relating Custom Property Changes to Instrument Script Execution

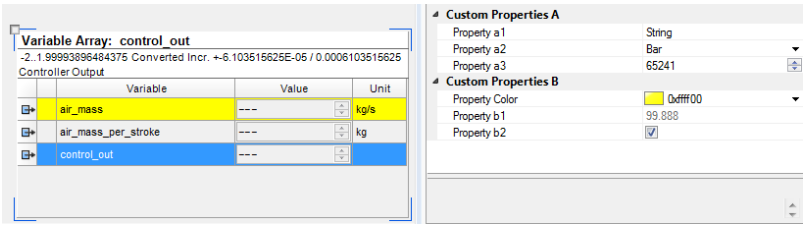
Introduction

In an instrument script you can specify code that is executed when a custom property is changed.

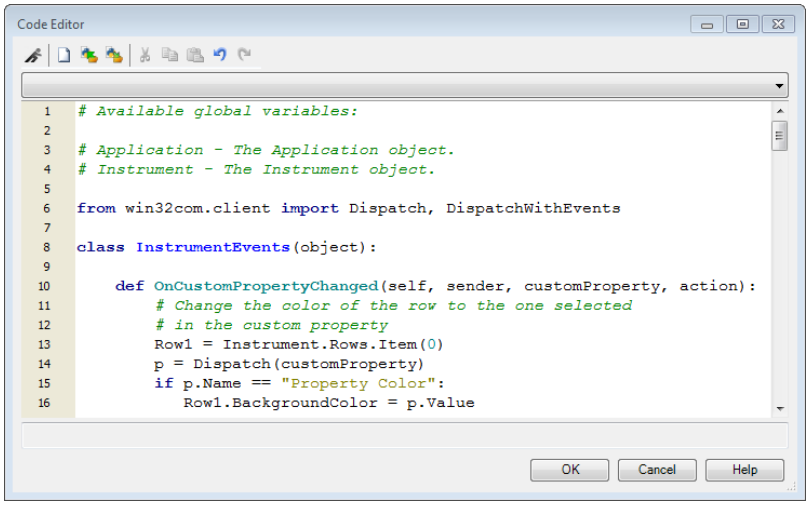
Using a custom property in an Instrument script

In the following example, the color of the first row in a Variable Array is changed when Property Color custom property is changed.

The following example shows a Variable Array with a custom property named Property Color.



The Python code in the instrument script is executed each time the value of Property Color changes. In this case, the color of the first row of the Variable Array is set to the value of Property Color.



For detailed information on the related instrument event, refer to:

- InstrumentCoreEvents / IViInstrumentCoreEvents <<EventInterface>> (ControlDesk Automation 🕮)
- CustomPropertyChangedAction <<Enumeration>> (ControlDesk Automation 🕮)

Related topics

References

CustomProperties / IViCustomProperties <<Interface>> (ControlDesk Automation 🕮)
CustomPropertyChangedAction <<Enumeration>> (ControlDesk Automation 🕮)
InstrumentCoreEvents / IViInstrumentCoreEvents <<EventInterface>> (ControlDesk Automation 🕮)

Utilities and Software Demos

Where to go from here

Information in this section

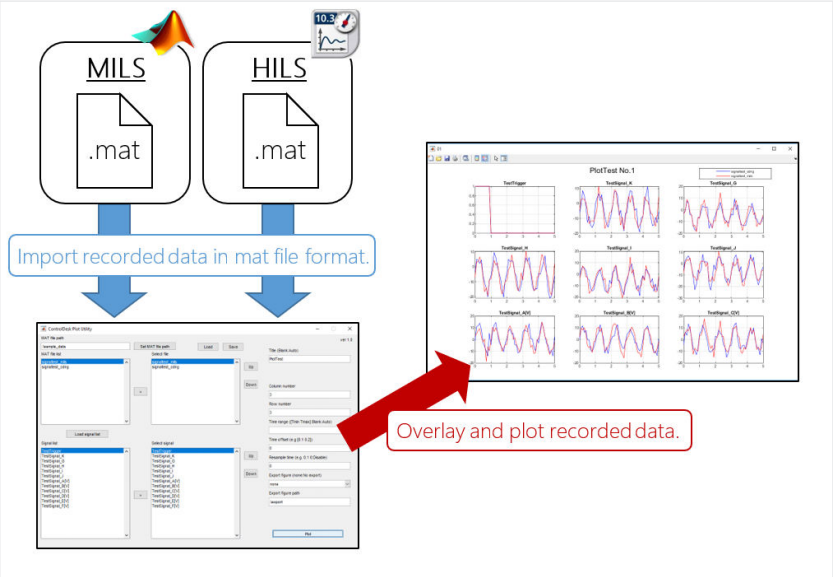
ControlDesk Plot Utility

Introduction

This utility is a MATLAB® script that lets you compare measurement data from different sources, such as MATLAB® simulations or ControlDesk recordings.

Description

The utility lets you import MAT files with measurement data from different sources, and display the differences in a MATLAB plot chart. For example, you can compare results of model-in-the-loop (MIL) simulations recorded in MATLAB and results of hardware-in-the-loop (HIL) simulations (HILS) on a dSPACE real-time simulator recorded with ControlDesk.



Availability of the utility

The utility is available via the dSPACE website. Refer to http://www.dspace.com/go/utility_cd_plot.

Find Variable on Layouts Utility

Introduction

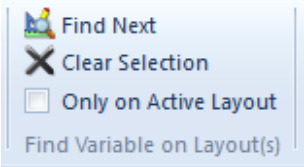
This utility finds all instruments that are connected to a selected variable.

Description

This utility adds a new ribbon group to the Layouting ribbon and a new command to the context menu of the Variables controlbar.

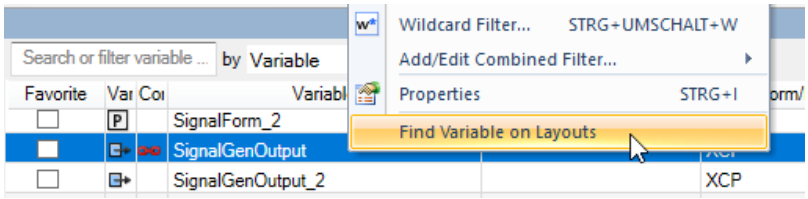
Ribbon group The new Find Variable on Layouts ribbon group provides the following three elements:

- Find Next
- Clear Selection
- Only on active layout



Context menu The context menu of the Variables controlbar provides the following new commands:

- Find Variable on Layouts
- Find Variable on Active Layout (if the Only on active layout checkbox is selected in the ribbon group)



Availability of the utility

The utility is available via the dSPACE website. Refer to http://www.dspace.com/go/utility_cd_find_var_on_lay.

Related topics

Basics

Basics of Placing Variables on a Layout (ControlDesk Layouting 🕮)
Basics on Customizing Ribbons via Extension Scripts 16
Extending the Context Menu of Elements 27

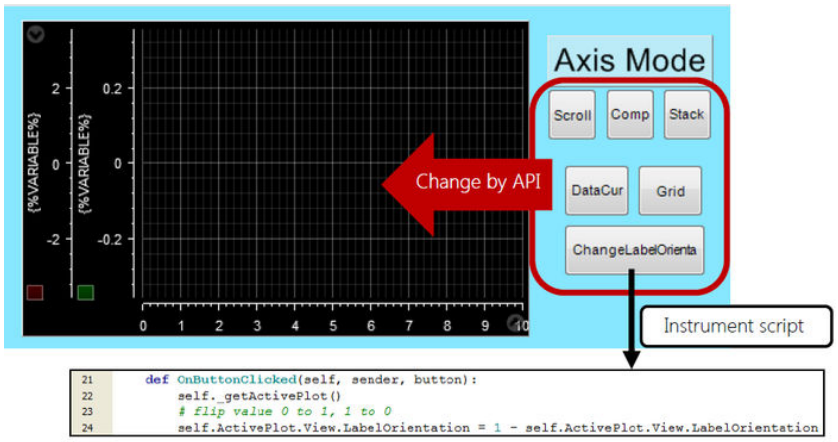
Functional Time Plotter Instrument Demo

Introduction

This software demo offers a Time Plotter with buttons for changing the display settings.

Description

- You can change the display settings of the Time Plotter via the buttons.
- You can customize the instrument by changing the instrument script of each button.



Availability of the software demo

The software demo is available via the dSPACE website. Refer to http://www.dspace.com/go/swdemo_cd_func_timeplt.

Related topics

Basics

Adding a Python Script to an Instrument or Layout.....37
Basics of Handling the Time Plotter (ControlDesk Instrument Handling 🕮)

HowTos

How to Optimize the Instrument Arrangement (ControlDesk Layouting 🕮)

References

Push Button (ControlDesk Instrument Handling 🕮)

Measurement State Change Instruments Demo

Introduction

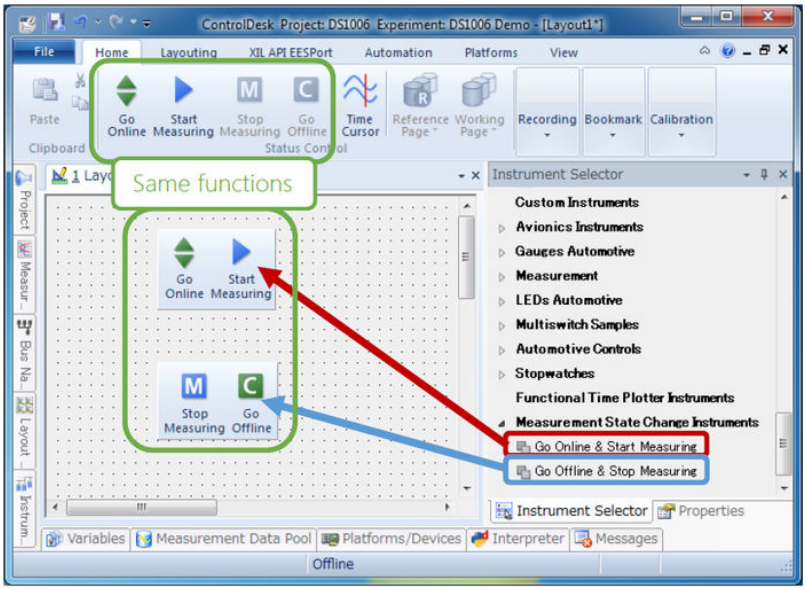
This software demo offers two instruments that let you go online or offline and start or stop a measurement, even if ControlDesk is in full-screen mode.

Description

The new instruments offer the following commands:

- Go Online
- Go Offline
- Start Measuring
- Stop Measuring

The instruments let you use these commands even if ControlDesk is in full-screen mode.



Availability of the software demo

The software demo is available via the dSPACE website. Refer to http://www.dspace.com/go/swdemo_cd_state_change_instr.

Related topics

Basics

Basics of Handling the Invisible Switch (ControlDesk Instrument Handling 🕮)

HowTos

How to Start Measuring (ControlDesk Measurement and Recording 🕮)
How to Stop Measuring (ControlDesk Measurement and Recording 🕮)

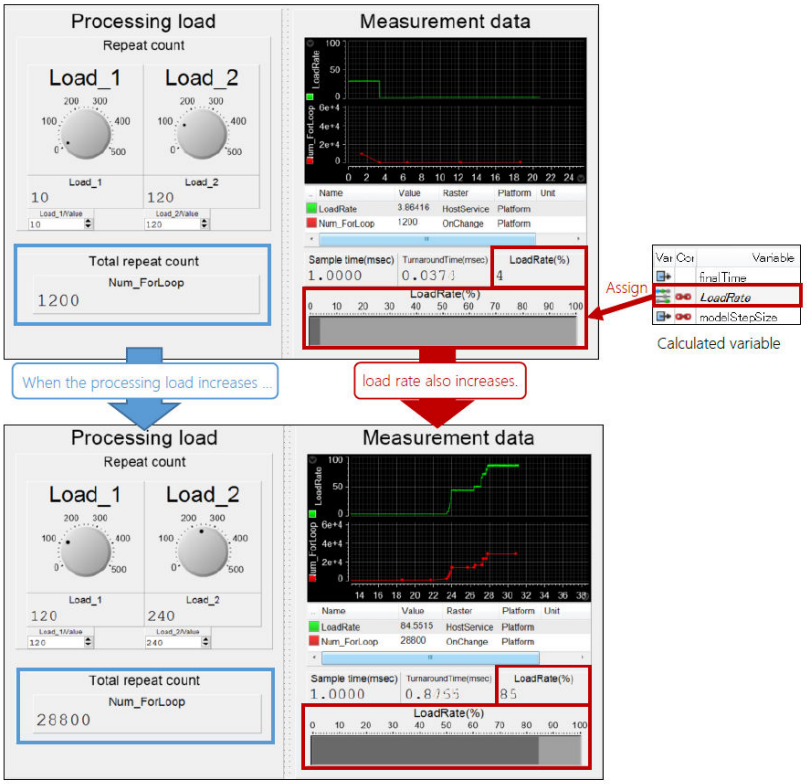
Processing Load Rate Value Demo

Introduction

This software demo provides a calculated variable that computes the load rate of a platform.

Description

The following illustration shows an example of using the LoadRate variable in instruments.



Availability of the software demo

The software demo is available via the dSPACE website. Refer to http://www.dspace.com/go/swdemo_cd_load_rate_val.

Related topics

Basics

Basics on Defining Calculated Variables (ControlDesk Variable Management 🕮)

Replace/Reload Data Sets Utility

Introduction

If you have stored CDFX data sets or sub data sets in the file system, this utility extends the possibilities to use them in ControlDesk.

- You can replace data sets in ControlDesk with data sets from the file system.

- You can reload data sets in ControlDesk if you have imported them from the file system.
- When you start online calibration, ControlDesk checks the originally imported data set files of involved data sets for changes.

Description

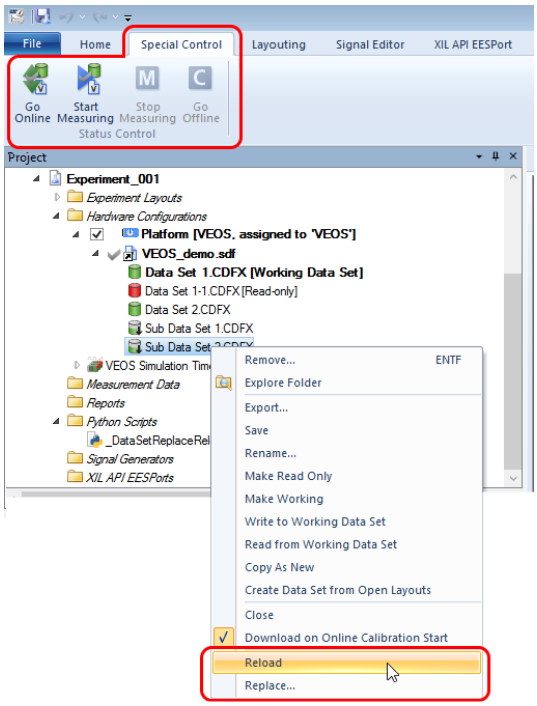
This utility adds new commands to the context menu of a data set and to the ControlDesk ribbon.

Context menu The context menu of a data set is extended by the following commands.

- Replace
- Reload

Ribbon The ControlDesk ribbon is extended by the Special Control ribbon that contains commands that include a check of the originally imported files.

- Go Online
- Start Measuring



Availability of the utility

The utility is available via the dSPACE website. Refer to http://www.dspace.com/go/utility_cd_rep_rel_ds.

Related topics

Basics

Basics of Data Sets (ControlDesk Calibration and Data Set Management 🕮)
Basics on Customizing Ribbons via Extension Scripts..... 16
Extending the Context Menu of Elements..... 27

HowTos

How to Select Additional Data Sets for Downloading on Online Calibration Start (ControlDesk Calibration and Data Set Management 🕮)

Glossary

Introduction

Briefly explains the most important expressions and naming conventions used in the ControlDesk documentation.

Where to go from here

Information in this section

Numerics

3-D Viewer An instrument for displaying items in a 3-D environment.

A

A2L file A file that contains all the relevant information on measurement and calibration variables in an ECU application and the ECU's communication interface(s). This includes information on the variables' memory addresses and conversion methods, the memory layout and data structures in the ECU as well as interface description data (IF_DATA).

Acquisition An object in the Measurement Configuration controlbar that specifies the variables to be measured and their measurement configuration.

Active variable description The variable description that is currently active for a platform/device. Multiple variable descriptions can be assigned to one platform/device, but only one of them can be active at a time.

Additional write variable A scalar parameter or writable measurement variable that can be connected to an instrument in addition to the main variable. When the value of the main variable changes, the changed value is also applied to all the additional write variables connected to the instrument.

Airspeed Indicator An instrument for displaying the airspeed of a simulated aircraft.

Altimeter An instrument for displaying the altitude of a simulated aircraft.

Animated Needle An instrument for displaying the value of a connected variable by a needle deflection.

Application image An image file that contains all the files that are created when the user builds a real-time application. It particularly includes the variable

description (SDF) file. To extend a real-time application, ControlDesk lets the user create an updated application image from a data set. The updated application image then contains a real-time application with an additional set of parameter values.

Artificial Horizon An instrument displaying the rotation on both the lateral and the longitudinal axis to indicate the angle of pitch and roll of a simulated aircraft. The Artificial Horizon has a pitch scale and a roll scale.

Automatic Reconnect Feature for automatically reconnecting to platform/device hardware, for example, when the ignition is turned off and on, or when the physical connection between the ControlDesk PC and the ECU is temporarily interrupted.

If the feature is enabled for a platform/device and if the platform/device is in the 'unplugged' state, ControlDesk tries to re-establish the logical connection to the platform/device hardware. After the logical connection is re-established, the platform/device has the same state as before the unplugged state was detected. A measurement started before the unplugged state was detected is resumed.

Automation A communication mechanism that can be used by various programming languages. A client can use it to control a server by calling methods and properties of the server's automation interface.

Automation script A script that uses automation to control an automation server.

Axis point object Common axis

B

Bar An instrument (or a value cell type of the Variable Array) for displaying a numerical value as a bar deflection on a horizontal or vertical scale.

Bitfield A value cell type of the Variable Array for displaying and editing the source value of a parameter as a bit string.

Bookmark A marker for a certain event during a measurement or recording.

Browser An instrument for displaying HTML and TXT files. It also supports Microsoft Internet Explorer© plug-ins that are installed on your system.

Bus communication replay A feature of the Bus Navigator that lets you replay logged bus communication data from a log file. You can add replay nodes

to the Bus Navigator tree for this purpose. You can specify filters to replay selected parts of the logged bus communication.

Bus configuration A configuration of all the controllers, communication matrices, and messages/frames/PDUs of a specific communication bus such as CAN. ControlDesk lets you display and experiment with bus configurations in the Bus Navigator.

Bus connection A mode for connecting dSPACE real-time hardware to the host PC via bus. The list below shows the possible bus connections:

- dSPACE real-time hardware installed directly in the host PC
- dSPACE real-time hardware installed in an expansion box connected to the host PC via dSPACE link board

Bus Instrument An instrument available for the Bus Navigator. It can be configured for different purposes, for example, to display information on received messages (RX messages) or to manipulate and transmit messages (TX messages). The instrument is tailor-made and displays only the message- and signal-specific settings which are enabled for display and/or manipulation by ControlDesk during run time.

Bus logging A feature of the Bus Navigator that lets you log raw bus communication data. You can add logger nodes on different hierarchy levels of the Bus Navigator tree for this purpose. You can specify filters to log filtered bus communication. The logged bus communication can be replayed.

Bus monitoring A feature of the Bus Navigator that lets you observe bus communication. You can open monitoring lists and add monitor nodes on different hierarchy levels of the Bus Navigator tree for this purpose. You can specify filters to monitor filtered bus communication.

Bus Navigator A controlbar for handling bus messages, such as CAN messages, LIN frames, and Ethernet packets.

Bus statistics A feature of the Bus Navigator that lets you display and log statistical information on the bus load during bus monitoring.

Bypassing A method for replacing an existing ECU function by running a new function.

C

Calculated variable A scalar variable that can be measured and recorded, and that is derived from one or more *input variables*.

The following input variable types are supported:

- Measurement variables
- Single elements of measurement arrays or value blocks
- Scalar parameters, or existing calculated variables

The value of a calculated variable is calculated via a user-defined *computation formula* that uses one or more input variables.

Calculated variables are represented by the  symbol.

CalDemo ECU A demo program that runs on the same PC as ControlDesk. It simulates an ECU on which the Universal Measurement and Calibration (XCP) protocol and the Unified Diagnostic Services (UDS) protocol are implemented.

The CalDemo ECU allows you to perform parameter calibration, variable measurement, and ECU diagnostics with ControlDesk under realistic conditions, but without having to have a real ECU connected to the PC. Communication between the CalDemo ECU and ControlDesk can be established via XCP on CAN or XCP on Ethernet, and UDS on CAN.

Tip

If communication is established via XCP on Ethernet, the CalDemo ECU can also run on a PC different from the PC on which ControlDesk is running.

The memory of the CalDemo ECU consists of two areas called memory page. Each page contains a complete set of parameters, but only one page is accessible by the CalDemo ECU at a time. You can easily switch the memory pages of the CalDemo ECU to change from one parameter to another in a single step.

Two ECU tasks run on the CalDemo ECU:

- ECU task #1 runs at a fixed sample time of 5 ms. In ControlDesk's **Measurement Configuration**, ECU task #1 is related to the time-based **5 ms**, **10 ms**, **50 ms** and **100 ms** measurement rasters of the CalDemo ECU.
- ECU task #2 has a variable sample time. Whenever the CalDemo ECU program is started, the initial sample time is 5 ms. This can then be increased or decreased by using the **dSPACE CalDemo** dialog.

  ECU task #2 is related to the **extEvent** measurement raster of the CalDemo ECU.

The CalDemo ECU can also be used to execute diagnostic services and jobs, handle DTCs and perform measurement and calibration via ECU diagnostics.

The CalDemo ECU program is run by invoking `CalDemo.exe`. The file is located in the `.\Demos\CalDemo` folder of the ControlDesk installation.

Calibration Changing the parameter values of real-time applications or ECU applications.

Calibration memory segment Part of the memory of an ECU containing the calibratable parameters. Memory segments can be defined as `MEMORY_SEGMENT` in the A2L file. ControlDesk can use the segments to evaluate the memory pages of the ECU.

ControlDesk lets you perform the calibration of:

- Parameters inside memory segments
- Parameters outside memory segments
- Parameters even if no memory segments are defined in the A2L file.

CAN Bus Monitoring device A device that monitors the data stream on a CAN bus connected to the ControlDesk PC.

The CAN Bus Monitoring device works, for example, with PC-based CAN interfaces such as the DCI-CAN2 or the DCI-CAN/LIN1.

The device supports the following variable description file types:

- DBC
- FIBEX
- AUTOSAR system description (ARXML)

CANGenerator A demo program that simulates a CAN system, that is, it generates signals that can be measured and recorded with ControlDesk. The program runs on the same PC as ControlDesk.

The CANGenerator allows you to use the CAN Bus Monitoring device under realistic conditions, but without having to have any device hardware connected to the PC.

The CAN (Controller Area Network) protocol is used for communication between the CANGenerator and ControlDesk. However, since the CANGenerator runs on the same PC as ControlDesk, ControlDesk does not communicate with the device via a real CAN channel, but via a *virtual CAN channel* implemented on the host PC.

You can start the CAN generator program by running `CANGenerator.exe`. The file is located in the `.\Demos\CANGenerator` folder of the ControlDesk installation.

Capture A data packet of all the measurement variables assigned to a measurement raster. The packet comprises the data that results from a single triggering of the raster.

CCP Abbreviation of CAN Calibration Protocol. This protocol can be implemented on electronic control units (ECUs) and allows users to access ECUs with measurement and calibration systems (MCS) such as ControlDesk.

The basic features of CCP are:

- Read and write access to the ECU memory, i.e., providing access for calibration
- Synchronous data acquisition
- Flash programming for ECU development purposes

The CCP protocol was developed by ASAM e.V. (Association for Standardization of Automation and Measuring Systems e.V.). For the protocol specification, refer to http://www.asam.net.

The following device supports ECUs with an integrated CCP service:

- CCP device

CCP device A device that provides access to an ECU with CCP connected to the ControlDesk PC via CAN, for example, for measurement and calibration purposes via CCP (CAN Calibration Protocol).

Check Button An instrument (or a cell type of the Variable Array) for displaying whether the value of a connected variable matches predefined values or for writing a predefined value to a connected variable.

cmdloader A command line tool for handling applications without using the user interface of an experiment software.

Common axis A parameter that consists of a 1-dimensional array containing axis points. A common axis can be referenced by one or more curves and/or maps. Calibrating the data points of a common axis affects all the curves and/or maps referencing the axis.

Common axes are represented by the symbol.

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>`

Computation method A formula or a table that defines the transformation of a source value into a converted value (and vice versa). In addition to the computation methods defined in the variable description file, ControlDesk provides the *__Identity* computation method which means the converted and the source value are equal.

Connected A platform/device state defined by the following characteristics:

- A continuous logical connection is established between ControlDesk and the platform/device hardware.
- A platform/device must be in the 'connected' state before it can change to the 'measuring/recording' or 'online calibration started' state.
- Online calibration is impossible. ControlDesk did not yet adjust the memory segments containing calibration data in the platform/device and on the corresponding hardware. Offline calibration is possible.
- Platform/device configuration is not possible. However, you can invoke platform/device configuration for a platform/device that is in the connected state. ControlDesk temporarily sets the platform/device to the disconnected state.

The 'connected' platform/device state is indicated by the icon.

Connection mode dSPACE real-time systems can be installed within the host PC or connected to the host via a bus interface and/or via Ethernet. When the Ethernet is being used, different network clients might exist. The connection type being used and, in the case of Ethernet, the network client being used, determine the dSPACE systems that can be accessed.

Control primitive A special diagnostic communication object for changing communication states or protocol parameters, or for identifying (ECU) variants.

Controlbar A window or pane outside the working area. Can be docked to an edge of the main window or float in front of it. A controlbar can contain a

document, such as a layout, or a tool, such as the **Bus Navigator**. It can be grouped with other controlbars in a window with tabbed pages.

ControlDesk The main version of ControlDesk for creating and running experiments, and for accessing dSPACE real-time hardware and VEOS. The functionality can be extended by optional software modules.

ControlDesk - Operator Version A version of ControlDesk that provides only a subset of functionality for running existing experiments. The functionality can be extended by optional software modules.

ControlDesk Bus Navigator Module An optional software module for ControlDesk for handling bus messages, such as CAN, LIN, and FlexRay messages, frames, and PDUs and Ethernet packets.

ControlDesk ECU Diagnostics Module An optional software module for ControlDesk that facilitates the calibration and validation of ECU diagnostic functions.

ControlDesk ECU Interface Module An optional software module for ControlDesk for calibration and measurement access to electronic control units (ECUs). The module is also required for calibration and measurement access to virtual ECUs (V-ECUs) used in SIL testing scenarios.

ControlDesk Signal Editor Module An optional software module for ControlDesk for the graphical definition and execution of signal generators for stimulating model variables of real-time/offline simulation applications.

Controller board Single-board hardware computing the real-time application. Contains a real-time processor for fast calculation of the model and I/O interfaces for carrying out the control developments.

Conversion table A table that specifies the value conversion of a source value into a converted value. In the case of verbal conversion, the converted value is a string that represents one numerical value or a range of numerical values.

Conversion type The type of a computation method, for example a linear function or a verbal computation method.

Curve A parameter that consists of

- A 1-dimensional array containing the axis points for the x-axis. This array can also be specified by a reference to a common axis.
- Another 1-dimensional array containing data points. The curve assigns one data point to each axis point.

Curves are represented by the  symbol.

D

DAQ module A hardware module for the acquisition of physical quantities

Data Cursor One or two cursors that are used to display the values of selected chart positions in a Time Plotter or an Index Plotter.

Data logger An object in the Measurement Configuration controlbar that lets you configure a data logging.

Data logger signal list A list that contains the variables to be included in subsequent data loggings on real-time hardware.

Data logging The recording of data on dSPACE real-time hardware that does not require a physical connection between the host PC and the real-time hardware. In contrast to flight recording, data logging is configured in ControlDesk.

Data set A set of the parameters and their values of a platform/device derived from the variable description of the platform/device. There are different types of data sets:

- Reference data set
- Sub data set
- Unassigned data set
- Working data set

DCI-CAN/LIN1 A dSPACE-specific interface between the host PC and the CAN/CAN FD bus and/or LIN bus. The DCI-CAN/LIN1 transfers messages between the CAN-/LIN-based devices and the host PC via the universal serial bus (USB).

DCI-CAN2 A dSPACE-specific interface between the host PC and the CAN bus. The DCI-CAN2 transfers CAN and CAN FD messages between the CAN-based devices and the host PC via the universal serial bus (USB).

DCI-GSI2 Abbreviation of *dSPACE Communication Interface - Generic Serial Interface 2*. A dSPACE-specific interface for ECU calibration, measurement and ECU interfacing.

DCI-GSI2 device A device that provides access to an ECU with DCI-GSI2 connected to the ControlDesk PC for measurement, calibration, and bypassing purposes via the ECU's debug interface.

DCI-KLine1 Abbreviation of *dSPACE Communication Interface - K-Line Interface*. A dSPACE-specific interface between the host PC and the diagnostics bus via K-Line.

Debug interface An ECU interface for diagnostics tasks and flashing.

Default raster A platform-/device-specific measurement raster that is used when a variable of the platform/device is connected to a plotter or a recorder, for example.

Deposition definition A definition specifying the sequence in which the axis point values of a curve or map are deposited in memory.

Device A software component for carrying out calibration and/or measurement, bypassing, ECU flash programming, or ECU diagnostics tasks.

ControlDesk provides the following devices:

- Bus devices:
  - CAN Bus Monitoring device
  - Ethernet Bus Monitoring device
  - LIN Bus Monitoring device
- ECU Diagnostics device
- GNSS device
- Measurement and calibration devices:
  - CCP device
  - DCI-GSI2 device
  - XCP on CAN device
  - XCP on Ethernet device

Each device usually has a variable description that specifies the device's variables to be calibrated and measured.

Diagnostic interface Interface for accessing the fault memory of an ECU.

Diagnostic job (often called Java job) Programmed sequence that is usually built from a sequence of the diagnostic service. A diagnostic job is either a single-ECU job or a multiple-ECU job, depending on whether it communicates with one ECU or multiple ECUs.

Diagnostic protocol A protocol that defines how an ECU communicates with a connected diagnostic tester. The protocol must be implemented on the ECU and on the tester. The diagnostics database specifies the diagnostic protocol(s) supported by a specific ECU.

ControlDesk's ECU Diagnostics device supports CAN and K-Line as the physical layers for communication with an ECU connected to the ControlDesk PC. For information on the supported diagnostic protocols with CAN and K-Line, refer to Basics of ECU Diagnostics with ControlDesk (ControlDesk ECU Diagnostics 🕮).

Diagnostic service A service implemented on the ECU as a basic diagnostic communication element. Communication is performed by selecting a service, configuring its parameters, executing it, and receiving the ECU results. When a service is executed, a defined request is sent to the ECU and the ECU answers with a specific response.

Diagnostic trouble code (DTC) A hexadecimal index for the identification of vehicle malfunctions. DTCs are stored in the fault memory of ECUs and can be read by diagnostic testers.

Diagnostics database A database that completely describes one or more ECUs with respect to diagnostics communication. ControlDesk supports the ASAM MCD-2 D ODX database format, which was standardized by ASAM e.V. (Association for Standardisation of Automation and Measuring Systems e.V.). For the format specification, refer to http://www.asam.net.

Proprietary diagnostics database formats are not supported by ControlDesk.

Diagnostics Instrument An instrument for communicating with an ECU via the diagnostic protocol using diagnostic services, diagnostic jobs, and control primitives.

Disabled A platform/device state defined by the following characteristics:

- No logical connection is established between ControlDesk and the platform/device hardware.
- When a platform/device is disabled, ControlDesk does not try to establish the logical connection for that platform/device. Any communication between the platform/device hardware and ControlDesk is rejected.
- Online calibration is impossible. Offline calibration is possible.
- Platform/device configuration is possible.

The 'disabled' platform/device state is indicated by the icon.

Disconnected A platform/device state defined by the following characteristics:

- No logical connection is established between ControlDesk and the platform/device hardware.
- When a platform/device is in the disconnected state, ControlDesk does not try to re-establish the logical connection for that platform/device.
- Online calibration is impossible. Offline calibration is possible.
- Platform/device configuration is possible.

The 'disconnected' platform/device state is indicated by the icon.

Display An instrument (or a value cell type of the Variable Array) for displaying the value of a scalar variable or the text content of an ASCII variable.

Documents folder A standard folder for user-specific documents.

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

DS1006 Processor Board platform A platform that provides access to a DS1006 Processor Board connected to the host PC for HIL simulation and function prototyping purposes.

DS1007 PPC Processor Board platform A platform that provides access to a single multicore DS1007 PPC Processor Board or a DS1007 multiprocessor system consisting of two or more DS1007 PPC Processor Boards, connected to the host PC for HIL simulation and function prototyping purposes.

DS1104 R&D Controller Board platform A platform that provides access to a DS1104 R&D Controller Board installed in the host PC for function prototyping purposes.

DS1202 MicroLabBox platform A platform that provides access to a MicroLabBox connected to the host PC for function prototyping purposes.

DsDAQ service A service in a real-time application or offline simulation application (OSA) that provides measurement data from the application to the

host PC. Unlike the host service, the DsDAQ service lets you perform, for example, triggered measurements with complex trigger conditions.

The following platforms support applications that contain the DsDAQ service:

- DS1007 PPC Processor Board platform
- DS1202 MicroLabBox platform
- MicroAutoBox III platform
- SCALEXIO platform
- VEOS platform
- XIL API MAPort platform

dSPACE Calibration and Bypassing Service An ECU service for measurement, calibration, bypassing, and ECU flash programming. The dSPACE Calibration and Bypassing Service can be integrated on the ECU. It provides access to the ECU application and the ECU resources and is used to control communication between an ECU and a calibration and/or bypassing tool.

With the dSPACE Calibration and Bypassing Service, users can run measurement, calibration, bypassing, and flash programming tasks on an ECU via the DCI-GSI2. The service is also designed for bypassing ECU functions using dSPACE prototyping hardware by means of the RTI Bypass Blockset in connection with DPMEM PODs. The dSPACE Calibration and Bypassing Service allows measurement, calibration, and bypassing tasks to be performed in parallel.

dSPACE Internal Bypassing Service An ECU service for on-target prototyping. The dSPACE Internal Bypassing Service can be integrated in the ECU application. It lets you add additional functions to be executed in the context of the ECU application without the need for recompiling the ECU application.

dSPACE Log A collection of errors, warnings, information, questions, and advice issued by all dSPACE products and connected systems over more than one session.

dSPACE system A hardware system such as a MicroAutoBox III or SCALEXIO system on which the real-time application runs.

Duration trigger A trigger that defines a duration. Using a duration trigger, you can, for example, specify the duration of data acquisition for a measurement raster. A duration trigger can be used as a stop trigger.

E

ECU Abbreviation of *electronic control unit*.

ECU application A sequence of operations executed by an ECU. An ECU application is mostly represented by a group of files such as ECU Image files, MAP files, A2L files and/or software module description files.

ECU calibration interface Interface for accessing an ECU by either emulating the ECU's memory or using a communication protocol (for example, XCP on CAN).

ECU diagnostics Functions such as:

- Handling the ECU fault memory: Entries in the ECU's fault memory can be read, cleared, and saved.
- Executing diagnostic services and jobs: Users can communicate with an ECU via a diagnostic protocol using diagnostic services, diagnostic jobs, and control primitives.

ControlDesk provides the ECU Diagnostics device device to access ECUs for diagnostic tasks. Communication is via diagnostic protocols implemented on the ECUs.

ECU diagnostics with ControlDesk are completely based on Open Diagnostic Data Exchange (ODX), the ASAM MCD-2 D diagnostics standard.

ControlDesk provides the Fault Memory Instrument and the Diagnostics Instrument for ECU diagnostics tasks.

ECU Diagnostics device A device that provides access to ECUs connected to the ControlDesk PC via CAN or K-Line for diagnostics or flash programming purposes.

ControlDesk provides the *ECU Diagnostics v2.0.2* device, which supports the ASAM MCD-3 D V2.0.2 standard.

ControlDesk supports the following ODX database standards:

- ASAM MCD-2 D V2.0.1
- ASAM MCD-2 D V2.2.0 (ISO 22901-1)

ECU flash programming A method by which new code or data is stored in ECU flash memory.

ECU Image file A binary file that is part of the ECU application. It usually contains the code of an ECU application and the data of the parameters within the application. It can be stored as an Intel Hex (HEX) or Motorola S-Record (MOT or S19) file.

EESPort Configurations controlbar A controlbar for configuring error configurations.

Electrical error simulation Simulating electrical errors such as loose contacts, broken cables, and short-circuits, in the wiring of an ECU. Electrical error simulation is performed by the failure simulation hardware of an HIL simulator.

Electrical Error Simulation port (EESPort) An *Electrical Error Simulation port* (EESPort) provides access to a failure simulation hardware for simulating electrical errors in an ECU wiring according to the ASAM AE XIL API standard.

The configuration of the EESPort is described by a hardware-dependent *port configuration* and one or more *error configurations*.

Environment model A model that represents a part or all of the ECU's environment in a simulation scenario.

The environment model is a part of the simulation system.

Environment VPU The executable of an environment model built for the VEOS platform. An environment VPU is part of an offline simulation application (OSA).

Error An electrical error that is specified by:

- An error category
- An error type
- A load type

Error category The error category defines how a signal is disturbed. Which errors you can create for a signal depends on the connected failure simulation hardware.

Error configuration An XML file that describes a sequence of errors you want to switch during electrical error simulation. Each error configuration comprises error sets with one or more errors.

Error set An error set is used to group errors (pin failures).

Error type The error type specifies the way an error category – i.e., an interruption or short circuit of signals – is provided. The error type defines the disturbance itself.

Ethernet Bus Monitoring device A device that monitors the data stream on an Ethernet network connected to the ControlDesk PC.

The device supports the following variable description file type:

- AUTOSAR system description (ARXML)

Ethernet connection A mode for connecting dSPACE real-time hardware to the host PC via Ethernet. The list below shows the possible Ethernet connections:

- dSPACE real-time hardware installed in an expansion box connected to the host PC via Ethernet.
- MicroAutoBox II/III and MicroLabBox connected via Ethernet.

Ethernet decoding A feature of the Bus Navigator that lets you view protocol data and raw data of an Ethernet frame.

Event An event that is triggered by an action performed in ControlDesk.

Event context The scope of validity of event sources and events. There is one event handler code area for each event context.

Event handler Code that is executed when the related event occurs.

Event management Functionality for executing custom code according to actions triggered by ControlDesk.

Event source An object providing and triggering events. *LayoutManagement* is an example of an event source.

Event state State of an event. ControlDesk provides the following event states:

- No event handler is defined
- Event handler is defined and enabled
- Event handler is defined and disabled
- Event handler is defined, but no Python code is available
- Event handler is deactivated because a run-time error occurred during the execution of the Python code

Expansion box A box that hosts dSPACE boards. It can be connected to the host PC via bus connection or via network.

Experiment A container for collecting and managing information and files required for a parameter calibration and/or measurement task. A number of experiments can be collected in a project but only one of them can be active.

Extension script A Python script (PY or PYC file) that is executed each time ControlDesk starts up. An extension script can be executed for all users or user-specifically.

F

Failure insertion unit Hardware unit used with dSPACE simulators to simulate failures in the wiring of an ECU, such as broken wire and short circuit to ground.

Fault memory Part of the ECU memory that stores diagnostic trouble code (DTC) entries with status and environment information.

Fault Memory Instrument An instrument for reading, clearing, and saving the content of the ECU's fault memory.

Firmware update An update for the firmware installed in the board's flash memory. Firmware should be updated if it is older than required by the real-time application to be downloaded.

Fixed axis An axis with data points that are not deposited in the ECU memory. Unlike a common axis, a fixed axis is specified within a curve or map. The parameters of a fixed axis cannot be calibrated.

Fixed parameter A parameter that has a fixed value during a running simulation. Changing the value of a fixed parameter does not immediately affect the simulation results. The affect occurs only after you stop the simulation and

start it again. A fixed parameter is represented by an added pin in its symbol, for example:  .

Flash job A specific diagnostic job for flashing the ECU memory. A flash job implements the process control for flashing the ECU memory, such as initialization, security access, writing data blocks, etc.

Flight recording The recording of data on dSPACE real-time hardware that does not require a physical connection between the host PC and the real-time hardware. In contrast to data logging, flight recording is not configured in ControlDesk but via RTI and RTLib.

Frame An instrument for adding a background frame to a layout, for example, to visualize an instrument group.

G

Gauge An instrument for displaying the value of the connected variable by a needle deflection on a circular scale.

Gigalink module A dSPACE board for connecting several processor boards in a multiprocessor system. The board allows high-speed serial data transmission via fiber-optic cable.

GNSS data Positioning and timing data that is transmitted by a Global Navigation Satellite System (GNSS), such as GPS, GLONASS, or Galileo. GNSS receivers use this data to determine their location.

GNSS device A device that provides positioning data from a GNSS receiver (e.g., a serial GPS mouse) in ControlDesk.

ControlDesk provides the *GNSS (GPS, GLONASS, Galileo, ...)* device that supports various global navigation satellite systems.

GPX file An XML file that contains geodata, such as waypoints, routes, or tracks. In ControlDesk, you can import GPX files to visualize GNSS positioning data in a Map instrument.

Group A collection of variables that are grouped according to a certain criterion.

H

Heading Indicator An instrument displaying the heading direction of a simulated aircraft on a circular scale.

Host service A service in a real-time application that provides measurement data from the application to the host PC.

The following platforms support applications that contain the host service:

- DS1006 Processor Board platform
- DS1104 R&D Controller Board platform
- MicroAutoBox platform
- Multiprocessor System platform

Index Plotter A plotter instrument for displaying signals that are measured in an event-based raster (index plots).

Input quantity A measurement variable that is referenced by a common axis and that provides the input value of that axis.

Instrument An on-screen representation that is designed to monitor and/or control simulator variables interactively and to display data captures. Instruments can be arranged freely on layouts.

The following instruments can be used in ControlDesk:

- 3-D Viewer
- Airspeed Indicator
- Altimeter
- Animated Needle
- Artificial Horizon
- Bar
- Browser
- Bus Instrument
- Check Button
- Diagnostics Instrument
- Display
- Fault Memory Instrument
- Frame
- Gauge
- Heading Indicator
- Index Plotter
- Invisible Switch
- Knob
- Multistate Display
- Multiswitch
- Numeric Input
- On/Off Button

- Push Button
- Radio Button
- Selection Box
- Slider
- Sound Controller
- Static Text
- Steering Controller
- Table Editor
- Time Plotter
- Variable Array
- XY Plotter

Instrument Navigator A controlbar that displays a tree with all the instruments of the active layout and all the variables that are connected to them. The Instrument Navigator's main function is easy selection of instruments in complex layouts.

Instrument script A Python script used to extend the functionality of an instrument.

Instrument Selector A controlbar that provides access to ControlDesk's instruments. The instruments can be placed on a layout via double-click or drag & drop.

Interface description data (IF_DATA) An information structure, mostly provided by an A2L file, describing the type, features and configuration of an implemented ECU interface.

Internal Interpreter ControlDesk's built-in programming interface for editing, running and importing Python scripts. It contains an Interpreter controlbar where the user can enter Python commands interactively and which displays output and error messages of Python commands.

Interpreter controlbar A controlbar that can be used to execute line-based commands. It is used by the Internal Interpreter to print out Python standard error messages and standard output during the execution or import of Python scripts.

Invisible Switch An instrument for defining an area that is sensitive to mouse operations.

IOCNET IOCNET (I/O carrier network) is a dSPACE-specific high-speed serial communication bus that connects all the real-time hardware in a SCALEXIO system. IOCNET can also be used to build a multiprocessor system that consists of multiple SCALEXIO processor hardware components.

K

Knob An instrument for displaying and setting the value of the connected variable by means of a knob on a circular scale.

L

Label list A list of user-defined variables that can be used for saving connected variables, etc.

Layout A window with instrument s connected to variables of one or more simulation models.

Layout Navigator A controlbar that displays all opened layout s. It can be used for switching between layouts.

Layout script A Python script used to extend the functionality of a layout .

Leading raster The measurement raster that specifies the trigger settings for the Time Plotter display. The leading raster determines the time range that is visible in the plotter if a start and stop trigger is used for displaying the signals.

LIN Bus Monitoring device A device that monitors the data stream on a LIN bus connected to the ControlDesk PC.

The LIN Bus Monitoring device works, for example, with PC-based LIN interfaces.

The device supports the following variable description file types:

- LDF
- FIBEX
- AUTOSAR system description (ARXML)

Load type The load type specifies the option to disturb a signal with or without load rejection.

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>`

Logical link A representation of an ECU specified in the diagnostics database. A logical link contains information on the ECU itself, and all the information required for accessing it, such as the diagnostic protocol used for

communication between the ECU and ControlDesk. Each logical link is represented by a unique short name in the ODX database.

Look-up table A look-up table maps one or more input values to one output value. You have to differentiate between the following look-up table types:

- A 1-D look-up table maps one input value to one output value.
- A 2-D look-up table maps two input values to one output value.
- An n-D look-up table maps multidimensional table data with 3 or more input values to one output value.

Look-up table is a generic term for curves and maps.

M

Main variable A scalar variable that is visualized in an instrument that can be used to change parameter values. In addition to the main variable, additional write variables can also be connected to (but not visualized in) the same instrument. When you change the value of the main variable in an instrument, the changed value is also applied to all the additional write variables connected to that instrument.

Map A parameter that consists of

- A 1-dimensional array containing the axis points for the x-axis. This array can also be specified by a reference to a common axis.
- A 1-dimensional array containing the axis points for the y-axis. This array can also be specified by a reference to a common axis.
- A 2–dimensional array containing data points. The map assigns one data point of the array to each pair of x-axis and y-axis points.

Maps are represented by the  symbol.

Map file A file that contains symbols (symbolic names) and their physical addresses. It is generated during a build process of an ECU application.

Map instrument A customized Browser instrument. It uses an instrument script to open a web map and connect positioning data to the map. The Map instrument offers prepared connection nodes to connect variables with GNSS data.

Measurement Viewing and analyzing the time traces of variables, for example, to observe the effects of ECU parameter changes.

ControlDesk provides various instruments for measuring variables.

Measurement (variable type) A scalar variable that can be measured, including individual elements of a measurement array.

Measurement variables are represented by the  symbol.

Measurement array A 1-, 2-, or 3-dimensional array of measurement variables. In variable lists, ControlDesk displays entries for the measurement array itself and for each array element.

Measurement arrays are represented by the  symbol.

Measurement buffer A ring buffer that buffers measurement data at the start of a measurement. The measurement buffer size determines the amount of data that can be buffered. Earlier values are overwritten by later values when the buffer capacity is exceeded (buffer overflow).

Measurement Configuration A controlbar that allows you to configure measurement, recording and data logging.

Measurement Data API Application programming interface for accessing measurement data. The API lets the user access measurement data without having to use ControlDesk.

Measurement Data Pool A controlbar that provides access to measurement data recorded in measurement data files.

Measurement raster Specification of how often a value of a variable is updated during a measurement. A measurement raster can be derived from a measurement service.

Measurement service The generic term for the following services:

- CCP service
- DsDAQ service
- Host service
- XCP service

Measurement signal list A list containing the variables to be included in subsequent measurements and recording. The list is global for all platforms/devices of the current experiment. The measurement signal list is available in the configuration area of the **Measurement Configuration** controlbar.

Measurement variable Any variable type that can be measured but not calibrated.

Measuring/recording A platform/device state defined by the following characteristics:

- A continuous logical connection is established between ControlDesk and the platform/device hardware.
- Online calibration is possible. Parameter values can be changed directly on the platform/device hardware.
- A measurement (or recording) is running.
- Platform/device configuration is not possible.

The 'measuring' / 'recording' platform/device state is indicated by the  icon.

Memory page An area of a calibration memory. Each page contains a complete set of parameters of the platform/device hardware, but only one of the pages is "visible" to the microcontroller of the ECU or the real-time processor (RTP) of the platform hardware at a time.

ControlDesk supports platform/device hardware with up to two memory pages. These are usually the working page and the reference page. The parameter values on the two memory pages usually are different. ControlDesk lets you switch from one page to the other, so that when parameters are changed on one page, the changes can be made available to the ECU or prototyping hardware via a single page switch.

Messages controlbar A controlbar displaying a history of all error and warning messages that occur during work with ControlDesk.

MicroAutoBox III platform A platform that provides access to a MicroAutoBox III connected to the host PC for function prototyping purposes such as Bypassing.

MicroAutoBox platform A platform that provides access to a MicroAutoBox II connected to the host PC for function prototyping purposes such as bypassing.

Mirrored memory A memory area created by ControlDesk on the host PC that mirrors the contents of the available memory pages of calibration and prototyping hardware. For hardware with two memory pages, the mirrored memory is divided into a reference and a working page, each of them containing a complete set of parameters. When a calibration or prototyping platform/device is added to an experiment, ControlDesk initially fills the available memory pages of the mirrored memory with the contents of the ECU Image file (initial filling for calibration devices) or with the contents of the SDF file (initial filling for platforms).

- Mirrored memory for offline calibration

  Parameter values can even be changed offline. Changes to parameter values that are made offline affect only the mirrored memory.
- Offline-to-online transition for online calibration

  For online calibration, an offline-to-online transition must be performed. During the transition, ControlDesk compares the memory pages of the hardware of each platform/device with the corresponding pages of the mirrored memory. If the pages differ, the user has to equalize them by uploading them from the hardware to the host PC, or downloading them from the host PC to the hardware.
- Mirrored memory for online calibration

  When ControlDesk is in the online mode, parameter value changes become effective synchronously on the memory pages of the hardware and in the mirrored memory. In other words, parameter values on the hardware and on the host PC are always the same while you are performing online calibration.

Modular system A dSPACE processor board and one or more I/O boards connected to it.

Multi-capture history The storage of all the captures acquired during a triggered measurement. The amount of stored data depends on the measurement buffer.

Multi-pin error A feature of the SCALEXIO concept for electrical error simulation that lets you simulate a short circuit between three or more signal

channels and/or bus channels. The channels can be located on the same or different boards or I/O units. You can simulate a short circuit between:

- Channels of the same signal category (e.g., four signal generation channels)
- Channels of different signal categories (e.g., three signal generation channels and two signal measurement channels)
- Signal channels and bus channels (e.g., two signal generation channels, one signal measurement channel, and one bus channel)

Multiple electrical errors A feature of the SCALEXIO concept for electrical error simulation that lets you switch electrical errors at the same time or in succession. For example, you can simulate an open circuit for one channel and a short circuit for another channel at the same time, without deactivating the first error.

Multiprocessor System platform A platform that provides access to:

- A multicore application running on a multicore DS1006 board
- A multiprocessor application on a multiprocessor system consisting of two or more DS1006 processor boards interconnected via Gigalink.

ControlDesk handles a multiprocessor/multicore system as a unit and uses one system description file (SDF file) to load the applications to all the processor boards/cores in the system.

Multistate Display An instrument for displaying the value of a variable as an LED state and/or as a message text.

Multistate LED A value cell type of the Variable Array for displaying the value of a variable as an LED state.

Multiswitch An instrument for changing variable values by clicking sensitive areas in the instrument and for visualizing different states depending on the current value of the connected variable.

N

Numeric Input An instrument (or a value cell type of the Variable Array) for displaying and setting the value of the connected variable numerically.

O

Observing variables Reading variable values cyclically from the dSPACE real-time hardware and displaying their current values in ControlDesk, even if no measurement is running. Variable observation is performed without using a measurement buffer, and no value history is kept.

For platforms that support variable observation, variable observation is available for parameters and measurement variables that are visualized in single-shot instruments (all instruments except for a plotter). If you visualize a variable in a single-shot instrument, the variable is not added to the measurement signal list. Visualizing a parameter or measurement variable in a plotter automatically adds the variable to the measurement signal list.

ControlDesk starts observing variables if one of the following conditions is true:

- Online Calibration is started for the platform.

  All the parameters and measurement variables that are visualized in single-shot instruments are observed.
- Measurement is started for the platform.

  All the visualized parameters and measurement variables that are not activated for measurement in the measurement signal list are observed. Data of the activated parameters and measurement variables is acquired using measurement rasters.

ODX database Abbreviation of Open Diagnostic Data Exchange, a diagnostics database that is the central ECU description for working with an ECU Diagnostics device in ControlDesk. The ODX database contains all the information required to perform diagnostic communication between ControlDesk and a specific ECU or set of ECUs in a vehicle network. ControlDesk expects the database to be compliant with ASAM MCD-2 D (ODX).

Offline State in which the parameter values of platform/device hardware in the current experiment cannot be changed. This applies regardless of whether or not the host PC is physically connected to the hardware.

The mirrored memory allows parameter values to be changed even offline.

Offline simulation A PC-based simulation in which the simulator is not connected to a physical system and is thus independent of the real time.

Offline simulation application (OSA) An offline simulation application (OSA) file is an executable file for VEOS. After the build process with a tool such as the VEOS Player, the OSA file can be downloaded to VEOS.

An OSA contains one or more VPUs, such as V-ECUs and/or environment VPUs.

On/Off Button An instrument (or a value cell type of the Variable Array) for setting the value of the connected parameter to a predefined value when the button is pressed (On value) and released (Off value).

Online calibration started A platform/device state defined by the following characteristics:

- A continuous logical connection is established between ControlDesk and the platform/device hardware.
- Online calibration is possible. Parameter values can be changed directly on the platform/device hardware.
- Platform/device configuration is not possible.

Before starting online calibration, ControlDesk lets you compare the memory pages on the platform/device hardware with the corresponding pages of the mirrored memory. If the parameter values on the pages differ, they must be

equalized by uploading the values from the hardware to ControlDesk, or downloading the values from ControlDesk to the hardware. However, a page cannot be downloaded if it is read-only.

The 'online calibration started' platform/device state is indicated by the symbol.

Operation signal A signal which represents the result of an arithmetical operation (such as addition or multiplication) between two other signals.

Operator mode A working mode of ControlDesk in which only a subset of the ControlDesk functionality is provided. You can work with existing experiments but not modify them, which protects them from unintentional changes.

Output parameter A parameter or writable measurement whose memory address is used to write the computed value of a calculated variable to.

P

Parameter Any variable type that can be calibrated.

Parameter (variable type) A scalar parameter, as well as the individual elements of a value block.

Scalar parameters are represented by the P symbol.

Parameter limits Limits within which parameters can be changed. Parameters have hard and weak limits.

- Hard limits

  Hard limits designate the value range of a parameter that you *cannot* cross during calibration.

  The hard limits of a parameter originate from the corresponding variable description and cannot be edited in ControlDesk.

- Weak limits

  Weak limits designate the value range of a parameter that you *should not* cross during calibration. When you cross the value range defined by the weak limits, ControlDesk warns you.

  In ControlDesk, you can edit the weak limits of a parameter within the value range given by the parameter's hard limits.

PHS (Peripheral High Speed) bus A dSPACE-specific bus for communication between a processor board and the I/O boards in a modular system. It allows direct I/O operations between the processor board (bus master) and I/O boards (bus slaves).

PHS-bus-based system A modular dSPACE system consisting of a processor board such as the DS1006 Processor Board and I/O boards. They communicate with each other via the PHS (Peripheral High Speed) bus.

Pitch variable A variable connected to the pitch scale of an Artificial Horizon.

Platform A software component representing a simulator where a simulation application is computed in real-time (on dSPACE real-time hardware) or in non-real-time (on VEOS).

ControlDesk provides the following platforms:

- DS1006 Processor Board platform
- DS1007 PPC Processor Board platform
- DS1104 R&D Controller Board platform
- DS1202 MicroLabBox platform
- MicroAutoBox platform
- MicroAutoBox III platform
- Multiprocessor System platform
- SCALEXIO platform
- VEOS platform
- XIL API MAPort platform

Each platform usually has a variable description that specifies its variables.

Platform trigger A trigger that is available for a platform and that is evaluated on the related dSPACE real-time hardware or VEOS.

Platforms/Devices controlbar A controlbar that provides functions to handle devices, platforms, and the applications assigned to the platforms.

Plotter instrument ControlDesk offers three plotter instruments with different main purposes:

- The Index Plotter displays signals in relation to events.
- The Time Plotter displays signals in relation to measurement time.
- The XY Plotter displays signals in relation to other signals.

Port configuration To interface the failure simulation hardware, an EESPort needs the hardware-dependent *port configuration file* (PORTCONFIG file). The file's contents must fit the connected HIL simulator architecture and its failure simulation hardware.

Postprocessing The handling of measured and recorded data by the following actions:

- Displaying measured or recorded data
- Zooming into measured or recorded signals with a plotter
- Displaying the values of measurement variables and parameters as they were at any specific point in time

Processor board A board that computes real-time applications. It has an operating system that controls all calculations and communication to other boards.

Project A container for collecting and managing the information and files required for experiment/calibration/modification tasks in a number of experiments. A project collects the experiments and manages their common data.

Project controlbar A controlbar that provides access to projects and experiments and all the files they contain.

Project root directory The directory on your file system to which ControlDesk saves all the experiments and documents of a project. Every project is associated with a project root directory, and several projects can use the same project root directory. The user can group projects by specifying several project root directories.

ControlDesk uses the Documents folder as the default project root directory unless a different one is specified.

Properties controlbar A controlbar providing access to the properties of, for example, platforms/devices, layouts/instruments, and measurement/recording configurations.

Proposed calibration A calibration mode in which the parameter value changes that the user makes do not become effective on the hardware until they are applied. This allows several parameter changes to be written to the hardware together. Being in proposed calibration mode is like being in the offline calibration mode temporarily.

Push Button An instrument (or a value cell type of the Variable Array) for setting the value of the connected parameter by push buttons.

Python Editor An editor for opening and editing PY files.

Q

Quick start measurement A type of measurement in which all the ECU variables configured for measurement are measured and recorded, starting with the first execution of an ECU task. ControlDesk supports quick start measurements on ECUs with DCI-GSI2, CCP, and XCP (except for XCP on Ethernet with the TCP transmission protocol).

Quick start measurement can be used to perform cold start measurements. Cold start means that the vehicle and/or the engine are cooled down to the temperature of the environment and then started. One reason for performing cold start measurements is to observe the behavior of an engine during the warm-up phase.

R

Radio Button An instrument for displaying and setting the value of the connected parameter by radio buttons.

Real-time application An application that can be executed in real time on dSPACE real-time hardware. A real-time application can be built from a Simulink model containing RTI blocks, for example.

Record layout A record layout is used to specify a data type and define the order of the data in the memory of the target system (ECU, for example). For scalar data types, a record layout allows you to add an address mode (direct or indirect). For structured (aggregated) data types, the record layout specifies all the structure elements and the order they appear in.

The `RECORD_LAYOUT` keyword in an A2L file is used to specify the various record layouts of the data types in the memory. The structural setup of the various data types must be described in such a way that a standard application system will be able to process all data types (reading, writing, operating point display etc.).

Record layout component A component of a record layout. A structured record layout consists of several components according to the ASAP2 specification. For example, the AXIS_PTS_X component specifies the x-axis points, and the FNC_VALUES component describes the function values of a map or a curve.

Recorder An object in the Measurement Configuration controlbar that specifies and executes the recording of variables according to a specific measurement configuration.

Recorder signal list A list that contains the variables to be included in subsequent recordings.

Recording Saving the time traces of variables to a file. Both measurement variables and parameters can be recorded. Recorded data can be postprocessed directly in ControlDesk.

A recording can be started and stopped immediately or via a trigger:

- Immediate recording

  The recording is started and stopped without delay, without having to meet a trigger condition.
- Triggered recording

  The recording is not started or stopped until certain trigger conditions are met. These conditions can be defined and edited in ControlDesk.

Reduction data Additional content in an MF4 file that allows for visualizing the MF4 file data depending on the visualization resolution. Reduction data therefore improves the performance of the visualization and postprocessing of measurement data.

Reference data set A read-only data set assigned to the reference page of a device that has two memory pages. There can be only one reference data set for each device. The reference data set is read-only.

Reference page Memory area containing the parameters of an ECU. The reference page contains the read-only reference data set.

Note

Some platforms/devices provide only a working page. You cannot switch to a reference page in this case.

Resynchronization Mechanism to periodically synchronize the drifting timers of the platform/device hardware ControlDesk is connected to. Resynchronization means adjustment to a common time base.

Roll variable A variable connected to the roll scale of an Artificial Horizon.

S

Sample count trigger A trigger that specifies the number of samples in a data capture.

A sample count trigger can be used as a stop trigger.

SCALEXIO platform A platform that provides access to a single-core, multicore or multiprocessor SCALEXIO system connected to the host PC for HIL simulation and function prototyping purposes.

SCALEXIO system A dSPACE hardware-in-the-loop (HIL) system consisting of at least one processing hardware component, I/O boards, and I/O units. They communicate with each other via the IOCNET. In a SCALEXIO system, two types of processing hardware can be used, a DS6001 Processor Board or a real-time industry PC as the SCALEXIO Processing Unit. The SCALEXIO system simulates the environment to test an ECU. It provides the sensor signals for the ECU, measures the signals of the ECU, and provides the power (battery voltage) for the ECU and a bus interface for restbus simulation.

SDF file The system description file that describes the files to be loaded to the individual processing units of a simulation platform. It also contains the variable description of the relevant simulation application.

The SDF file is generated automatically when the TRC file is built.

Segment The minimum part a segment signal can consist of.

There are different kinds of segments to be used in segment signals:

- Segments to form synthetic signal shapes (sine, sawtooth, ramp, etc.)
- Segments to perform arithmetical operations (addition, multiplication) with other segments
- Segments to represent numerical signal data (measured data)

Segment signal A signal consisting of one or more segments.

Selection Box An instrument for selecting a text-value entry and setting the respective numerical value for the connected variable.

Signal

- Representation of a variable measured in a specific measurement raster.
- Generic term for segment signals and operation signals.

  A signal is part of a signal description set which can be displayed and edited in the working area.

Signal description set A group of one or more signals.

A signal description set and its signals can be edited in the working area by means of the Signal Editor. Each signal description set is stored as an STZ file either in the **Signal Description Sets** folder or in the **Signal Generators** folder.

Signal Editor A software component to create, configure, display, and manage signals in signal description sets.

Signal file A file that contains the wiring information of a simulator and that is part of the standard dSPACE documentation of dSPACE Simulator Full-Size. Normally, dSPACE generates this file when designing the simulator. Before using a failure simulation system, users can adapt the signal file to their needs.

Signal generator An STZ file containing a signal description set and optional information about the signal mapping, the description of variables, and the real-time platform.

The file is located in the **Signal Generators** folder and used to generate, download, and control Real-Time Testing sequences, which are executed on the real-time platform to stimulate model variables in real time.

Signal Mapping A controlbar of the Signal Editor to map model variables to signals and variable aliases of a signal generator.

Signal Selector A controlbar of the Signal Editor. The Signal Selector provides signals and segments for arranging and configuring signal description sets in the working area.

SIL testing Abbreviation of *software-in-the-loop testing*.

Simulation and testing of individual software functions, complete virtual ECUs (V-ECUs), or even V-ECU networks on a local PC or highly parallel in the cloud independently of real-time constraints and real hardware.

Simulation application The generic term for offline simulation application (OSA) and real-time application.

Simulation system A description of the composition of V-ECU models, environment models, real ECUs, and their interconnections required for simulating the behavior of a system. A simulation system is the basis for the generation of a simulation application for a given simulator platform.

Simulation time group Group of platforms/devices in an experiment whose simulation times are synchronized with each other. If resynchronization is enabled, ControlDesk synchronizes a simulation time group as a whole, not the single members of the group individually.

Simulator A system that imitates the characteristics or behaviors of a selected physical or abstract system.

Single-processor system A system that is based on one dSPACE processor or controller board.

Single-shot instrument An instrument that displays an instantaneous value of a connected variable without keeping a value history. In ControlDesk, all instruments except for a plotter are single-shot instruments. For platforms that support the variable observer functionality, you can use single-shot instruments to observe variables.

Slave application An application assigned to the slave DSP of a controller or I/O board. It is usually loaded and started together with the real-time application running on the corresponding main board.

Slave DSP A DSP subsystem installed on a controller or I/O board. Its slave application can be loaded together with the real-time application or separately.

Slider An instrument (or a value cell type of the Variable Array) for displaying and setting the value of the connected variable by means of a slide.

Sound Controller An instrument for generating sounds to be played.

Standard axis An axis with data points that are deposited in the ECU memory. Unlike a common axis, a standard axis is specified within a curve or map. The parameters of a standard axis can be calibrated, which affects only the related curve or map.

Start trigger A trigger that is used, for example, to start a measurement raster. A platform trigger can be used as a start trigger.

Static Text An instrument for displaying explanations or inscriptions on the layout.

Steering Controller An instrument for changing variable values using a game controller device such as a joystick or a steering wheel.

Stimulation Writing signals to variables in real-time models during a simulation run.

Stop trigger A trigger that is used, for example, to stop a measurement raster.

String A text variable in ASCII format.

Strings are represented by the  symbol.

Struct A variable with the struct data type. A struct contains a structured list of variables that can have various data types. In ControlDesk, a struct variable can contain either parameters and value blocks or measurement variables and measurement arrays. ControlDesk supports nested structs, i.e., structs that contain further structs and struct arrays as elements.

Structs are represented by the symbol.

Struct array An array of homogeneous struct variables.

Struct arrays are represented by the symbol.

STZ file A ZIP file containing signal descriptions in the STI format. The STZ file can also contain additional MAT files to describe numerical signal data.

Sub data set A data set that does not contain the complete set of the parameters of a platform/device.

Symbol A symbolic name of a physical address in a MAP file. A symbol can be associated to a variable in the Variable Editor, for example, to support an address updates.

System variable A type of variable that represents internal variables of the device or platform hardware and that can be used as measurement signals in ControlDesk to give feedback on the status of the related device or platform hardware. For example, an ECU's power supply status or the simulation state of a dSPACE board can be visualized via system variables.

T

Table Editor An instrument for displaying and setting values of a connected curve, map, value block, or axis in a 2-D, 3-D, and grid view. The Table Editor can also display the values of a measurement array.

The Table Editor can be used for the following variable types:

- Common axis ( )
- Curve ( )
- Map ( )
- Measurement array ( )
- Value block ( )

Time cursor A cursor which is visible at the same time position in the following instruments:

- In all Time Plotters
- In all XY Plotters
- In all bus monitoring lists

You can use the time cursor to view signal values at a specific point in time. If you move the time cursor, all measured signals and the respective parameters are

updated. Instruments and bus monitoring lists display the values that are available at the selected time position.

Time Plotter A plotter instrument for displaying signals that are measured in a time-based raster (time plots).

Topology A description of the processor boards belonging to a multiprocessor system and their interconnections via Gigalinks. The topology also contains information on which Gigalink port of each processor board is connected to the Gigalink ports of other processor boards in the multiprocessor system.

Topology information is contained in the real-time application (PPC/x86/RTA) files of the multiprocessor system's processor boards.

TRC file A variable description file with information on the variables available in an environment model running on a dSPACE platform.

Trigger A condition for executing an action such as starting and stopping a measurement raster or a recorder.

The generic term for the following trigger types:

- Duration trigger
- Platform trigger
- Sample count trigger

Trigger condition A formula that specifies the condition of a trigger mathematically.

Triggered measurement The measurement of a measurement raster started by a platform trigger. The data flow between the dSPACE real-time hardware or VEOS and the host PC is not continuous.

U

Unassigned data set A data set that is assigned neither to the working page nor to the reference page of a platform/device. An unassigned data set can be defined as the new working or reference data set. It then replaces the "old" working or reference data set and is written to the corresponding memory page, if one is available on the platform/device.

Unplugged A platform/device state defined by the following characteristics:

- The logical connection between ControlDesk and the hardware was interrupted, for example, because the ignition was turned off or the ControlDesk PC and the hardware were disconnected.
- Before the state of a platform/device changes to 'unplugged', the platform/device was in one of the following states:
  - 'Connected'
  - 'Online calibration started'
  - 'Measuring' / 'Recording'

Tip

A device for which the connection between ControlDesk and the device hardware currently is interrupted is also set to the 'unplugged' state when you start online calibration if both the following conditions are fulfilled:

- The device's **Start unplugged** property is enabled.
- The **Start online calibration behavior** property is set to 'Ignore differences'.

This is possible for CCP and XCP devices. For details on the two properties listed above, refer to General Settings Properties (ControlDesk Platform Management 🕮).

- If the Automatic Reconnect feature is enabled for a platform/device and if the platform/device is in the 'unplugged' state, ControlDesk periodically tries to re-establish the logical connection for that platform/device.
- Online calibration is impossible. Offline calibration is possible.
- Platform/device configuration is possible.

The 'unplugged' platform/device state is indicated by the  icon.

Untriggered measurement The measurement of a measurement raster not started by a platform trigger. The data flow between the dSPACE real-time hardware or VEOS and the host PC is continuous.

User function An external function or program that is added to the ControlDesk user interface for quick and easy access during work with ControlDesk.

User Functions Output A controlbar that provides access to the output of external tools added to the **Automation** ribbon.

V

Value block A parameter that consists of a 1- or 2-dimensional array of scalar parameters.

In variable lists, ControlDesk displays entries for the value block itself and for each array element.

Value blocks are represented by the  symbol.

Value conversion The conversion of the original *source values* of variables of an application running on an ECU or dSPACE real-time hardware into the corresponding scaled *converted values*.

Variable Any parameter or measurement variable defined in a variable description. ControlDesk provides various instruments to visualize variables.

Variable alias An alias name that lets the user control the property of a segment by a model parameter of a real-time application.

Variable Array An instrument for calibrating parameters and displaying measurement variable values.

The Variable Array can be used for the following variable types:

- Measurement ()
- Measurement array ()
- String ()
- Struct ()
- Struct array ()
- Value ()
- Value block ( )

Variable connection The connection of a variable to an instrument. Via the variable connection, data is exchanged between a variable and the instrument used to measure or calibrate the variable. In other words, variable connections are required to visualize variables in instrument.

Variable description A file describing the variables in a simulation application, which are available for measurement, calibration, and stimulation.

Variable Editor A tool for viewing, editing, and creating variable descriptions in the ASAM MCD-2MC (A2L) file format. The Variable Editor allows you to create A2L files from scratch, or to import existing A2L files for modification.

Variable Filter A variable filter contains the filter configuration of a combined filter, which is used to filter the variable list in the **Variables** controlbar using a combination of filter conditions.

Variables controlbar A controlbar that provides access to the variables of the currently open experiment.

V-ECU Abbreviation of *virtual ECU*.

ECU software that can be executed in a software-in-the-loop (SIL) testing environment such as a local PC or highly parallel in the cloud independently of real-time constraints and real ECU hardware.

Vehicle information The ODX database can contain information for one or more vehicles. Vehicle information data is used for vehicle identification purposes and for access to vehicles. It references the access paths (logical links) to the ECUs.

VEOS A simulator which is part of the PC and allows the user to run an offline simulation application (OSA) without relation to real time.

VEOS Player is the graphical user interface for VEOS.

VEOS platform A platform that configures and controls the offline simulation application (OSA) running in VEOS and that also provides access to the application's environment VPU.

VEOS Player An application running on the host PC for editing, configuring and controlling an offline simulation application (OSA) running in VEOS.

Verbal conversion A conversion in which a conversion table is used to specify the computation of numerical values into strings. The verbal conversion table is used when you switch the value representation from source to converted mode and vice versa.

Verbal conversion range A conversion in which a conversion table is used to specify the computation of a range of numerical values into strings. The verbal conversion range table is used when you switch the value representation from source to converted mode and vice versa.

View set A named configuration of the controlbars of ControlDesk. A view set has a default state and a current state that can differ from the default state. The configuration includes the geometry, visibility, and docking or floating state of controlbars.

Visualization The representation of variables in instruments:

- Measurement variables are visualized in instruments to view and analyze their time traces.
- Calibration parameters are visualized in instruments to change their values.

VPU Abbreviation of *virtual processing unit*. A VPU is part of an offline simulation application in VEOS. Each VPU runs in a separate process of the PC.

VPU is also the generic term for:

- V-ECUs
- Environment VPUs
- Controller VPUs
- Bus VPUs

W

Working data set The data set currently residing in the memory of a platform/device hardware. There can be only one working data set for each calibration platform/device. The working data set is read/write.

Working page Memory area containing the parameters of an ECU or prototyping hardware (memory page). The working page contains the read/write working data set.

If the platform/device also provides a reference page, ControlDesk lets you switch between both pages.

Writable measurement A scalar variable that can be measured and calibrated.

X

XCP Abbreviation of *Universal Measurement and Calibration Protocol*. A protocol that is implemented on electronic control units (ECUs) and provides access to ECUs with measurement and calibration systems (MCS) such as ControlDesk.

XCP is based on the *master-slave principle*:

- The ECU is the slave.
- The measurement and calibration system is the master.

The "X" stands for the physical layers for communication between the ECU and the MCS, such as CAN (Controller Area Network) and Ethernet.

The basic features of XCP are:

- ECU parameter calibration (CAL)
- Synchronous data acquisition (DAQ)
- Synchronous data stimulation (STIM), i.e., for bypassing
- ECU flash programming (PGM)

The XCP protocol was developed by ASAM e.V. (Association for Standardisation of Automation and Measuring Systems e.V.). For the protocol specification, refer to http://www.asam.net.

The following ControlDesk devices support ECUs with an integrated XCP service:

- XCP on CAN device
- XCP on Ethernet device

XCP on CAN device A device that provides access to an ECU with XCP connected to the ControlDesk PC via CAN. Using the XCP on CAN device, you can access the ECU for measurement and calibration purposes via XCP (*Universal Measurement and Calibration Protocol*).

XCP on Ethernet device A device that provides access to an ECU or V-ECU with XCP connected to the ControlDesk PC via Ethernet. The XCP on Ethernet device provides access to the ECU/V-ECU via XCP (*Universal Measurement and Calibration Protocol*) for measurement and calibration purposes.

XIL API EESPort Electrical Error Simulation port (EESPort)

XIL API MAPort platform A platform that provides access to a simulation platform via the ASAM XIL API implementation that is installed on your host PC.

XY Plotter A plotter instrument for displaying signals as functions of other signals.

A

B

C

D

E

I

L

M

P

S

U