Ansys

Guidelines

SCADE Test Automation Framework Guidelines

for ARINC 661-Compliant Systems

CONTACTS

Legal Contact

Ansys France
15 place Georges Pompidou
78180 Montigny-le-Bretonneux FRANCE
Phone: +33 1 30 60 15 00
Fax: +33 1 30 64 19 42

Technical Support

Ansys France
Parc Avenue, 9 rue Michel Labrousse
31100 Toulouse FRANCE
Phone: +33 5 34 60 90 50
Fax: +33 5 34 60 90 41

Submit questions to Ansys SCADE Products Technical Support at scade-support@ansys.com.

Contact one of our Sales representatives at scade-sales@ansys.com.

Direct general questions about SCADE products to scade-info@ansys.com.

Discover latest news on our products at www.ansys.com/products/embedded-software.

LEGAL INFORMATION

TERMS OF USE

DOCUMENTATION DISCLAIMER

THIRD-PARTY LEGAL INFO

The Legal Notice (PDF) about third-party software can be found under the help folder of the SCADE installation.

Shipping date: 17/10/24 **Revision**: ARC-TAF-EPG-25 - DOC/rev/63455-10

Guidelines Overview

These guidelines provide user-oriented instructions and technical reference information to help using and understanding SCADE Test Automation Framework. The document is divided into the following chapters:

- Chapter 1: "Introduction to SCADE Test Automation Framework"
- Chapter 2: "Setting Test Environment"
- Chapter 3: "Tutorial for Testing ToggleButton Widget"
- Chapter 4:"Using and Extending Widget Tests"

Appendixes and Index:

- Appendix A: "Testing Functions"
- Appendix B: "Coverage Functions"
- Appendix C: "A661 Widgets API"

"Index"

RELATED DOCUMENTS

- *SCADE Server Creator User Manual*

TYPOGRAPHICAL CONVENTIONS

Bold	GUI buttons, options, tabs, radio buttons, drop-down lists, window titles, panel names.
Italics	Path for menu selection from main menu bar or in contextual menus, new technical concepts or words, references to product documentation.
`Courier New`	Computer text, code extracts, constant or variable names, file names or directory path, typed text.
Title Case	Names of application components, menus.
UPPERCASE	File formats

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

1 / Introduction to SCADE Test Automation Framework

This chapter introduces the basic components, principles, and mechanisms that support SCADE Test Automation Framework.

Test Automation Framework Overview

SCADE Test Automation Framework provides comprehensive services allowing the testing automation of ARINC 661 widgets, considered as black box software components that are part of an ARINC 661-compliant graphic Cockpit Display System.

This framework aims at providing the foundations necessary to build your own widget test environment.

The following figure illustrates a possible test workflow using SCADE Test Automation Framework:

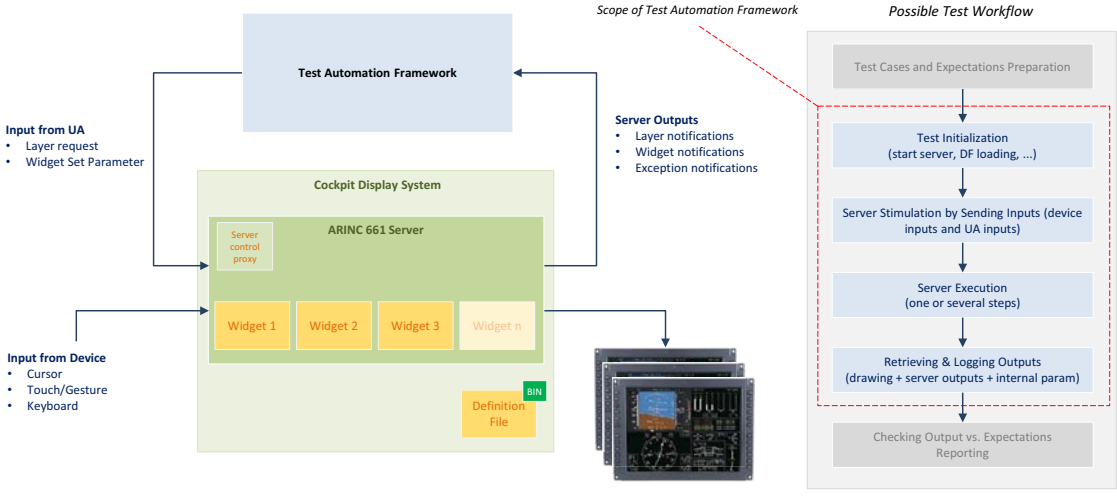


Figure 1.1: Example of a test workflow with SCADE Test Automation Framework

The Test Automation Framework is accessible in SCADE installation from the following directory: `\SCADE A661\PythonLib\taf`.

Test Automation Framework Architecture

SCADE Test Automation Framework uses a Server Control Proxy which is activated in the SCADE A661 Server to add monitoring and commanding capabilities (done via a dedicated A661 layer request and notification).

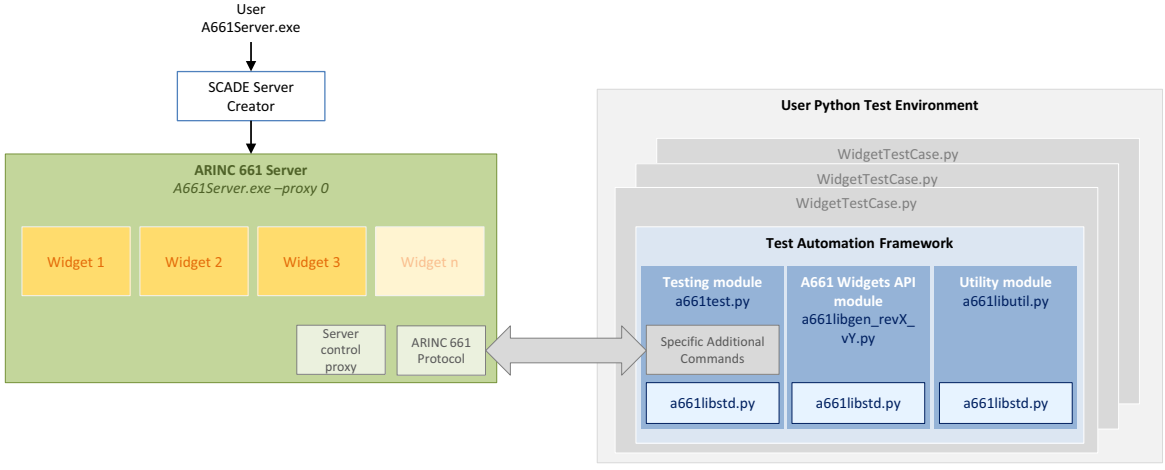


Figure 1.2: SCADE Test Automation Framework Architecture

SCADE Test Automation Framework includes a Python package with modules for testing and managing ARINC 661 objects like creating Definition File binaries, creating User Applications, or encoding/decoding ARINC 661 frames:

- **`a661libstd.py`**

  The `a661libstd.py` module provides Python classes representing the different A661 elements like server, widget, or command blocks. It contains also the server control proxy class used to communicate with the proxy on the server. It is imported by other modules of the package and is generally not imported directly by the test cases.

- **`a661test.py`**

  The `a661test.py` main testing module provides the TestFramework and the TAFCommandError classes. The TestFramework class is instantiated to create a test case, with all control and observation functions needed for the test (based on the server control proxy class and the widgets' API). This class contains also model coverage functions and other utilities. It imports the a661libstd module.

- **`a661testcheck.py`**

  The additional `a661testcheck.py` testing module provides check functions to use on retrieved widget data (A661 notifications, widget parameter, screenshot).

- **`a661libutil.py`**

  The `a661libutil.py` utility module is provided for generic services to manipulate ARINC 661 objects as ARINC 661 frames encoding and decoding.

- **`a661_mc_report.py`**

  The `a661_mc_report.py` module provides additional python functions to be used to produce the model coverage report.

- **`a661log.py`**

  An example of logging capabilities is provided in the `a661log.py` module, which enables console and file logging with basic formatting and filtering.

- **`a661libgenerator.py`**

  The `a661libgenerator.py` module is provided to generate the A661 Widgets API adapted to the version of the ARINC 661 Widget Library. By default, a version of the currently supported ARINC 661 supplement is provided. From the entry point of the SCADE ARINC 661 solution, an *a661.xml* file, it generates the *a661libgen_revX_vY* (X is the supplement; Y the library version).

- **`a661libgen_rev9_v1.py`**

  The version of the A661 Widgets API delivered with this release of the SCADE Test Automation Framework. It imports the a661libstd module.

Note

The generated A661 Widgets API module and the testing module work independently. Both reference the a661libstd module containing common and A661 version independent data and functions.

In case you need to test the widgets from a custom ARINC 661 Widget Library, it is necessary to regenerate the A661 Widgets API and use the server corresponding to this custom library version.

For more information about custom server generation, see "Configuring Server Generation" on page 84 of *SCADE Server Creator User Manual*.

---

2 / Setting Test Environment

This chapter describes how to set up the Python test environment to be able to use the Python package and prepare test cases.

Setup Overview

You set up the test environment by setting the Python environment and generating (if needed) the A661 Widgets API. Once the environment is set up, you can prepare your test case scripts.

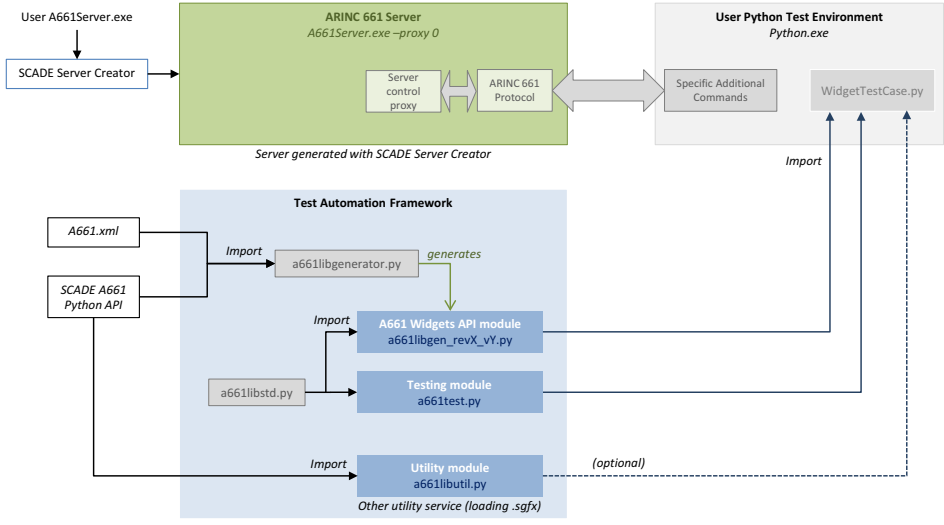


Figure 2.1: Test environment setup overview

You can read about all setup and preparation steps below:

- “Setting Up Python Environment”
- “Regenerating A661 Widgets API”
- “Preparing Test Cases”

Setting Up Python Environment

You must set the PYTHONPATH environment variable and install packages needed by SCADE Test Automation Framework.

Prerequisite: Python 3.6 or above (Python installation is provided at `<SCADE install>\SCADE\contrib\Python310`).

To set the Python environment

- Set the PYTHONPATH environment variable on the following directories:

`<SCADE install>\SCADE A661\PythonLib\`	A661 Test Automation Framework (**`taf`** package)
`<SCADE install>\SCADE\APIs\Python\lib\`	includes SCADE A661 Model API (SCADE package)

The `colorama` and `pillow` packages are needed (`pillow` is installed with the Python environment distributed with SCADE). You can install these packages with the following commands:

```
pip install colorama
pip install pillow
```

Traces are logged using the Python `logging` package. Each module of the **`taf`** package uses a logger named according to the module name. The a661log module allows to configure the **`taf`** main logger. For details, see "Logging Functions".

Regenerating A661 Widgets API

When the ARINC 661 Widget Library is customized, it is required to regenerate the A661 Widgets API Python module.

There are two options to generate this module.

- Access the Python shell and enter the following commands:

```
from taf.a661libgenerator import generate
generate('<path>/a661_description/a661.xml', '<gendir>')
```

- Open the command line and enter the following command:

```
python.exe <path>/a661libgenerator.py '<path>/a661_description/a661.xml' -o
'<gendir>'
```

GENERATED FILE

The generated file is named `a661libgen_revX_vY.py`, where `X` is the ARINC 661 supplement and `Y` the version of the ARINC 661 Widget Library.

Preparing Test Cases

Several steps need to be taken care of when you start writing your widget test cases. Please get acquainted with the following:

- “Importing Modules of “taf” Package”
- “Instantiating A661 Widgets API”
- “Instantiating TestFramework Class”

These steps are further illustrated when addressing a test case example in Chapter 3 about “Tutorial for Testing ToggleButton Widget”

Importing Modules of “taf” Package

Once the A661 Widgets API is generated (or if using the default SCADE A661 server), users need to import the Python modules (A661 Widgets API, testing, and utilities) for the test environment.

To import the modules

1 From a Python shell or in a script, use the following lines:

```
# The TAF commands module
from taf.a661test import *
# The A661 Widgets API module (or your own version of generated file)
from taf.a661libgen_revX_vY import *
```

where X is the ARINC 661 supplement and Y the version of the ARINC 661 Widget Library

2 If you want to import specific classes to use them in scripts, use the following lines:

```
from taf.a661test import <class_name>, …
from taf.a661libgen_rev9_v1 import <class_name>, …
```

Example:

```
from taf.a661test import TestFramework
from taf.a661libgen_rev9_v1 import Standard_rev9_v1, ToggleButton, Slider, Label
```

To use the logging functions provided with the **`taf`** package, the following line is required:

```
from taf.a661log import log_console, log_file
```

Optionally, you can import the `a661libutil.py` utility module for accessing additional utility functions (see “Utility Functions” on page 69).

Instantiating A661 Widgets API

You must first instantiate a StandardDescription class.

```
Standard_revX_vY()
```

where X is the ARINC 661 supplement and Y the version of the ARINC 661 Widget Library. By default, this is:

```
Standard_rev9_v1()
```

When a charset encoding block is defined in the DF and is different from A661_ASCII_EXTENDED, this charset encoding should be indicated when the class is instantiated. The argument 'utf-8' is passed for A661_UTF8, or 'gbk' for A661_GBK. For example:

```
Standard_rev9_v1('utf-8')
```

Instantiating TestFramework Class

After instantiating a StandardDescription class, you must instantiate an object of *TestFramework* class:

<te_obj> = TestFramework ()

Important

All commands used for Test Environment are methods of this *TestFramework* class. There are different ways to use the methods in Python. By default, the object of the *TestFramework* class is used to call each of the commands:

```
<te_obj>.<method>([<params>])
```

When an error is detected in any init, main, or coverage command, an exception `TAFCommandError` is raised with error id, error description, and name of the command as exception arguments.

The `TAFCommandError` exception class is defined in the same module as `TestFramework` class.

The exception is raised via the internal method `_handle_error` of the `TestFramework` class. Alternate error handling can be done by defining a child class of `TestFramework` on which this method is overridden.

3 / Tutorial for Testing ToggleButton Widget

This tutorial is a test case example that illustrates a possible usage of SCADE Test Automation Framework. **This tutorial is not intented as guidelines for testing widgets**.

In this tutorial, we will create a Python script to test the ToggleButton widget from the default ARINC 661 Widget Library provided in SCADE installation.

Before starting, make sure your Python environment is ready for using Test Automation Framework, as described in Chapter 2 about "Setting Test Environment".

Duration: 35-40 minutes

The tutorial is divided in two parts:

- "Creating Our First Test"
- "Adding Tests to Our Script"

Creating Our First Test

In the first part of this tutorial, we will define the different phases from the test initialization to the test execution and results verification.

Before starting test creation, we must create the following empty Python file `ToggleButtonTest.py`. When done, open the file in a text editor to create your first test as explained below.

Defining Setup Phase

First, we define a test setup phase with the following code. Copy these lines in the `ToggleButtonTest.py` file.

```
1     import logging
2     from taf.a661test import TestFramework, TAFCommandError
3     from taf.a661log import log_console
4     from taf.a661libgen_rev9_v1 import Standard_rev9_v1, ToggleButton
5     from taf.a661libutil import generate_binary_hex
6
7     Standard_rev9_v1()
8     te = TestFramework()
9
10    logger = logging.getLogger("taf")
11    log_console(True, "INFO")
12
13    try:
14       # DF Creation
15       # Server start and connection initialization
16       # Runtime phase
...
      except TAFCommandError as error:
          logger.error(f"Error {error.error} raised for command {error.command}: {error.description}")
          te.stop_server()
```

The setup phase consists of the following steps:

1. Import modules (see "Importing Modules of "taf" Package") with classes and functions necessary for this test:
   - *logging*: the python logging package
   - from *a661test* module, the *TestFramework* class containing all basic methods for testing and the *TAFCommandError* class for managing errors
   - from *a661log* module, the *log_console* function for displaying logs on console
   - from *a661libgen* module, the *Standard_rev9_v1* function for instantiating the main class of the A661 Widgets API and the *ToggleButton* class for accessing the widget to test
   - from *taf.a661libutil*, import `generate_binary_hex`
2. Instantiate the main class of the A661 Widgets API (see "Instantiating A661 Widgets API").
3. Instantiate an object of TestFramework class (see "Instantiating TestFramework Class").
4. Define the `taf` main logger and enable the console logging.

Note

To catch potential error and get feedback information, encapsulate all commands in a "try ... except" mechanism.

Creating Definition File

Now, we can create the definition file. The DF binary file is usually done using dfgen from a UAPC model. However, SCADE Test Automation Framework provides methods to produce this file directly in Python. Copy the following code (in bold) and paste it right after the initialization phase.

```
13    try:
14        # create a DF with a ToggleButton
15        df = te.create_df([ToggleButton(1).define(PosX=1000,PosY=1000)])
16        # save the DF as a binary file
17        generate_binary_hex(df, 'example1')
18
19    except TAFCommandError as error:
20        logger.error(f"Error {error.error} raised for command {error.command}: {error.description}")
21        te.stop_server()
```

DF Creation

The definition file creation consists of the following:

1. Create a DF for the ToggleButton: set the widget id (1) and its position in the DF. For brevity, our code is calling the create_df() and <widget>.define() functions with a limited number of arguments in the code snippet.

   There are additional arguments (see details in Appendix A about "Testing Functions"), but we let SCADE Test Automation Framework use the default values for those. Some of those key arguments and their default values:

   - create_df() function: ua_id = 1, layer_id = 1
   - <widget>.define() function: provided in the widget XML description file

   Note

   create_df() is used to create a basic DF object quickly, based on the Df class. You can create more complex DF objects by directly using this class. As documented in the related appendix, it is possible to use a list of widgets when you have several widgets. This would correspond to the content of a layer.

2. Generate the DF binary file. Once the DF object is created, the generate_binary_hex() method from the a661libutil module is used to generate the binary file.

Starting Server

The server is started with its DF binary file(s) and an optional configuration file, then the connection is initialized. Copy the following code (in bold) and paste it right after the definition file creation.

```
13   try:
14      # create a DF with a ToggleButton
15      df = te.create_df([ToggleButton(1).define(PosX=1000,PosY=1000)])
16      # save the DF as a binary file                                    Server Start
17      generate_binary_hex(df, 'example1')
18      # start the server with the DF as binary file
19      te.start_server("<SCADE_install>\\SCADE A661\\bin\\A661Server.exe", df_list=['example1.bin'])
20      # set connection with server
21      te.init_connection()
22      # register the Df object
23      te.register_df(df)
24
25   except TAFCommandError as error:
26       logger.error(f"Error {error.error} raised for command {error.command}: {error.description}")
27       te.stop_server()
```

In the above code snippet, replace <SCADE_install> with the path to your SCADE installation directory or replace the whole path to any place where the server to be used is located.

The start server phase consists of the following steps:

1. Start the server – provide server file path in your installation.
2. Initialize the connection.
3. Register the Df object to properly decode widget events.

For more details about functions used above, see “Start Server” on page 38, “Initialize Connection” on page 40, and “Register DF” on page 39.

Defining Runtime Phase

We can now write the runtime phase (added code in bold) and verify that, on start up, as layer 1 is active, the server generates the A661_NOTE_LAYER_IS_ACTIVE event for layer 1.

```
13    try:
14       # create a DF with a ToggleButton
15       df = te.create_df([ToggleButton(1).define(PosX=1000,PosY=1000)])
16       # save the DF as a binary file
17       generate_binary_hex(df, 'example1')
18       # start the server with the DF as binary file
19       te.start_server("<SCADE_install>\\SCADE A661\\bin\\A661Server.exe", df_list=['example1.bin'])
20       # set connection with server
21       te.init_connection()
22       # register the DF
23       te.register_df(df)
24       # server runtime phase: run one step to refresh the display
25       te.step(1)
26       # get all pending CDS notifications from the server
27       evt = te.pull_block()
28
29    except TAFCommandError as error:
30        logger.error(f"Error {error.error} raised for command {error.command}: {error.description}")
31        te.stop_server()
```

Runtime Phase

For our test, the runtime phase consists of the following steps:

1. Run the server for one execution step. As it is the first executed cycle, the content of the DU is displayed.
2. Call pull_block(). This method returns all pending notifications that are produced (here, the returned value is called `evt`). All other arguments are kept with default values.

First Test: Checking That Layer is Active

We can now verify that, upon startup, as layer 1 is active, the server generates the A661_NOTE_LAYER_IS_ACTIVE event for layer 1.

1 Import all the classes from the a661testcheck module (added code in bold).

```
1   import logging
2   from taf.a661test import TestFramework, TAFCommandError
3   from taf.a661log import log_console
4   from taf.a661libgen_rev9_v1 import Standard_rev9_v1, ToggleButton
5   from taf.a661libutil import generate_binary_hex
6   import taf.a661testcheck as checks
```

2 Call check_layer_event() to check that the returned event is the A661 event A661_NOTE_LAYER_IS_ACTIVE (added code in bold) .

```
14  try:
15     # create a DF with a ToggleButton
16     df = te.create_df([ToggleButton(1).define(PosX=1000,PosY=1000)])
17     # save the DF as a binary file
18     generate_binary_hex(df, 'example1')
19     # start the server with the DF as binary file
20     te.start_server("<SCADE_install>\\SCADE A661\\bin\\A661Server.exe", df_list=['example1.bin'])
21     # set connection with server
22     te.init_connection()
23     # register the DF
24     te.register_df(df)
25     # server runtime phase: run one step to refresh the display
26     te.step(1)
27     # get all pending CDS notifications from the server
28     evt = te.pull_block()
29     # check A661_NOTE_LAYER_IS_ACTIVE layer event
30     checks.check_layer_event(notifications=evt, event="A661_NOTE_LAYER_IS_ACTIVE")
31
32  except TAFCommandError as error:
33      logger.error(f"Error {error.error} raised for command {error.command}: {error.description}")
34      te.stop_server()
```

Returning Result Log

check_layer_event() returns a Boolean indicating whether the test is passed (true) or not (false). The Boolean can be used for logging or performing other actions outside the scope of this example.

1. Import the `colorama` module to receive colorized logging outputs.
2. Import all the classes from the **a661log** module (added code in bold).

```
1   import logging
2   import colorama
3   from taf.a661log import log_console, log_file
4   from taf.a661test import TestFramework, TAFCommandError
5   from taf.a661libgen_rev9_v1 import Standard_rev9_v1, ToggleButton
6   from taf.a661libutil import generate_binary_hex
7   import taf.a661testcheck as checks
8
9   Standard_rev9_v1()
10  te = TestFramework()
```

   The **a661log** module enables to configure the handlers for the console and file. It allows filtering the display of log levels and coloring logs depending on the log level.

3. Initialize the `colorama` module.
4. Set the log level to 'INFO' and filter this log level to get 'CHECK' information only (added code in bold).

```
1   import logging
2   import colorama
3   from taf.a661log import log_console, log_file
4   from taf.a661test import TestFramework, TAFCommandError
5   from taf.a661libgen_rev9_v1 import Standard_rev9_v1, ToggleButton
6   from taf.a661libutil import generate_binary_hex
7   import taf.a661testcheck as checks
8
9   Standard_rev9_v1()
10  te = TestFramework()
11
12  # Log level
13  logger = logging.getLogger("taf")
14  colorama.init()
15  log_console(True, "INFO",["CHECK"])
16  log_file(True,"<filepath>","INFO",["CHECK"])
```

Executing the Test

We can now execute our script. In a Windows Console, enter the following command:

```
python.exe ToggleButtonTest.py
```

You should obtain a message as below, meaning that the test passed:

```
C:\Working Directory\TestAutomationFramework>python toggle_button_test.py
CHECK OK> Layer 1/1 - Event A661_NOTE_LAYER_IS_ACTIVE - Correct received value: [0]
```

Congratulations! You have successfully created and run your first test with Test Automation Framework. You can continue with the second part of this tutorial to complete the script with additional tests.

If you wish to industrialize this code sample, you may adapt the above code to the Python test framework of your choice. The details are left as an exercise to the reader, but the first steps would be to split each of the above code snippets into its own function and to tie them together as required by the selected framework.

Adding Tests to Our Script

In the second part of this tutorial, we will add a test consisting in clicking on the ToggleButton and checking the result. Then, we will test a runtime message. Finally, we will modify a styleset at runtime.

Second Test: Clicking the ToggleButton

You will now add the definition of a test to click the button and change its state.

```
...
35      # click on ToggleButton and run until event is present (up to 10 cycles)
36      te.push_pointer(1100,1100,"LEFT","PRESSED",user_unit=True)
37      te.step(1)
38      # -1,-1 to keep same pointer position
39      te.push_pointer(-1,-1,"LEFT","RELEASED")
40      te.step(10,True)
41      # retrieve event and check that it is the expected one
42      evt = te.pull_block()
43      checks.check_widget_event(notifications=evt, event="A661_EVT_STATE_CHANGE", widget_id=1,
        expected_values="A661_SELECTED")
```

This test consists of the following steps:

5. Press the left button. It is pressed at location (1100, 1100) in user units, which is on the Toggle Button.
6. Execute 1 cycle.
7. Release the left button without moving the pointer (with first two arguments set to -1).
8. Execute 10 cycles until that notifications are produced.

   `until_has_blocks = True` means that the server is executed until an event is generated with a maximum number of cycles (here 10).

9 Call pull_block() to retrieve all the events that were produced.

10 Call check_widget_event() to verify that its state has changed.

The events were produced by pull_block().

The event name is as specified in the ARINC 661 Standard.

widget_id: previously, we created ToggleButton(1).

expected_values: event parameter as described in the ARINC 661 Standard.

We can now execute the updated script as explained in the first part of this tutorial.

As shown in the execution trace, step() waits until notifications are sent:

```
# LOAD_DF_INLINE
--> DF parsed for UA 1
# COMPLETE_DEFINITION
# CDS_STEP, nb=1, until_has_blocks=0
# PULL_BLOCK
# PUSH_POINTER_EVENT, dev_id=1
# CDS_STEP, nb=1, until_has_blocks=0
# PUSH_POINTER_EVENT, dev_id=1
# CDS_STEP, nb=10, until_has_blocks=1
# PULL_BLOCK
Connection for Test Automation Framework has been shut down
socket disconnected => server ended
Press any key to continue . . .
```

As displayed in the Console, the test should pass:

```
C:\Working Directory\TestAutomationFramework>python toggle_button_test.py
CHECK OK> Layer 1/1 - Event A661_NOTE_LAYER_IS_ACTIVE - Correct received value: [0]
CHECK OK> Widget 1/1/1 - Event A661_EVT_STATE_CHANGE - Correct received value: ButtonState=A661_SELECTED)
```

Third Test: Testing Runtime Messages

You will now add a third test in the script. We want to verify that the internal graphical state of the button is modified when the ARINC 661 parameter A661_INNER_STATE_TOGGLE is sent. For that purpose, move the cursor outside the button and send the runtime message.

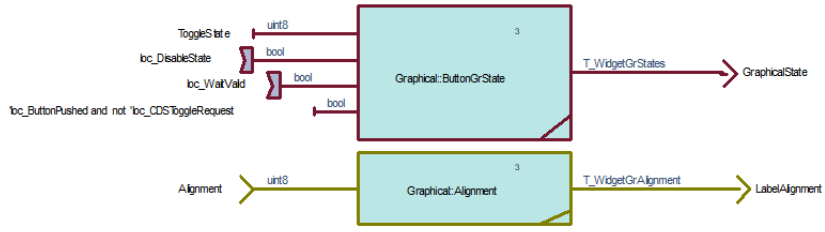
```
...
44      # move cursor outside ToggleButton
45      te.push_pointer(2000, 2000, user_unit = True)
46      # send runtime message to change button state
47      te.push_block(ToggleButton(1).inner_state_toggle("A661_UNSELECTED"))
48      # execute one step
49      te.step(1)
50      param = te.get_parameter(1, "GraphicalState")
51      #check that the graphical state is now Enabled+Unselected (1)
52      checks.check_parameter(param_value = param, expected_values = 1)
```

1. Move the cursor outside the Button.
2. Send the runtime message to modify the button state to Unselected.

   The name of the method is the name of the ARINC 661 message A661_INNER_STATE_TOGGLE without A661 and in small letters.

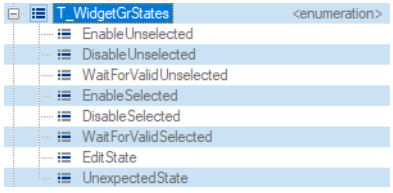
   The argument is the new state (here, the ARINC 661 constant name is used).
3. Execute 1 cycle.
4. Retrieve the state of the widget, by accessing to the corresponding implementation parameter "**GraphicalState**".

   This parameter is defined in the SCADE Suite behavior implementation of the widget. Note that it is not an A661 parameter.



5 Check that the value is **EnableUnselected** as in the enumerated type in the SCADE Suite behavior implementation.

expected_values: refers to the enumerated type in the SCADE Suite implementation. Its value must then be 1 (*i.e.*, the first value in the enumerated type with respect to Enum Value KCG pragma set to 1).



6 Execute the test.

Test execution result:

```
C:\Working Directory\TestAutomationFramework>python toggle_button_test.py
CHECK OK> Layer 1/1 - Event A661_NOTE_LAYER_IS_ACTIVE - Correct received value: [0]
CHECK OK> Widget 1/1/1 - Event A661_EVT_STATE_CHANGE - Correct received value: ButtonState=A661_SELECTED)
CHECK OK> Parameter value 1 retrieved as expected
```

If the test failed, the Console displays the message CHECK KO:

```
C:\Working Directory\TestAutomationFramework>python toggle_button_test.py
CHECK OK> Layer 1/1 - Event A661_NOTE_LAYER_IS_ACTIVE - Correct received value: [0]
CHECK OK> Widget 1/1/1 - Event A661_EVT_STATE_CHANGE - Correct received value: ButtonState=A661_SELECTED)
CHECK KO> Parameter value 1 does not correspond to any of expected ones ([2])
```

Fourth Test: Modifying Styleset at Runtime

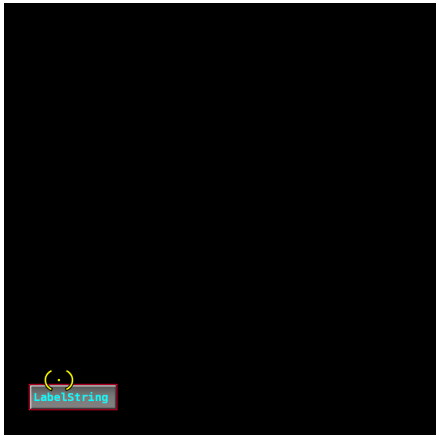
We will now add a last test to verify a styleset rendering. For this purpose, we will generate our own reference image file (ToggleButtonStyleset.png) and use it as the expected result.

```
...
53      # send runtime message to change the styleset
54      te.push_block(ToggleButton(1).style_set(111))
55      # execute one step
56      te.step(1)
57      # retrieve the produced image
58      screen_info=te.get_screenshot(resulting_file_path="myscreenshot")
59      #check that the screenshot is the expected one for this styleset
60      checks.check_image(actual_file=screen_info["file"],expected_file="ToggleButtonStyleset.png",
        save_to_file=True, imgdiff_dir="<SCADE_Installation>\\SCADE\\bin")
```

1. Send the runtime message to modify the styleset value using A661_STYLE_SET runtime parameter.
2. Run one cycle.
3. Retrieve the screenshot of the DU.

   get_screenshot() returns a dictionary composed of different elements including the pathname of the image file.
4. Verify that the graphical result is the expected one.
   - In *screen_info* structure, retrieve the screenshot file pathname.
   - *save_to_file*: when True, graphical differences are generated in the `comp_screenshots.bmp` file.
5. To compare the images, the executable SDYIMGDIFF is used. By default, this tool is stored in the SCADE bin directory. You can set the path for this tool or any other diff tool using the `imgdiff_dir` parameter.
6. Execute the test once to generate an PNG file named `myscreenshot-du01-18`.

During execution, the following screenshot is displayed on screen:



The test is failed as the specified image file does not exist:

```
C:\Working Directory\TestAutomationFramework>python toggle_button_test.py
CHECK OK> Layer 1/1 - Event A661_NOTE_LAYER_IS_ACTIVE - Correct received value: [0]
CHECK OK> Widget 1/1/1 - Event A661_EVT_STATE_CHANGE - Correct received value: ButtonState=A661_SELECTED)
CHECK OK> Parameter value 1 retrieved as expected
File error: file name or direct path of myscreenshot-du01-18.png actual image or ToggleButtonStyleset.png expected image
 is incorrect
CHECK KO> File error: file name or direct path of myscreenshot-du01-18.png actual image or ToggleButtonStyleset.png expe
cted image is incorrect
```

7 Rename the PNG file as `ToggleButtonStyleset.png`.

8 Execute the test again.

The displayed screenshot is the expected one. All tests are passed:

```
C:\Working Directory\TestAutomationFramework>python toggle_button_test.py
CHECK OK> Layer 1/1 - Event A661_NOTE_LAYER_IS_ACTIVE - Correct received value: [0]
CHECK OK> Widget 1/1/1 - Event A661_EVT_STATE_CHANGE - Correct received value: ButtonState=A661_SELECTED)
CHECK OK> Parameter value 1 retrieved as expected
CHECK OK> Screenshot myscreenshot-du01-18.png is identical to expected one ToggleButtonStyleset.png. Result of compariso
n is saved in file: comp_screenshots.bmp
```

Congratulations! You have finished this tutorial.

You have learned how to initialize, create, and run a test on a widget. You have learned how to test the widget at runtime and how to modify the information level of log results.

For further information about testing widgets with SCADE Test Automation Framework, refer to the following documentation:

- Appendix A about "Testing Functions": for a description of all the testing functions provided by the testing module
- Appendix C about "A661 Widgets API": for a description of all the classes and methods of the A661 Widgets API

4 / Using and Extending Widget Tests

In addition to the Test Automation Framework itself, the SCADE installation comes with a full implementation of a widget library test suite.

This chapter explains how CDS users that customize widgets can run the provided test suite on their own server project, to ensure that they do not break anything and to allow extension of the default tests for their own use.

Although the SCADE Test Automation Framework can be used in many different ways by Python developers to write custom tests, we propose our own way of running the SCADE Test Automation Framework in a self-contained and complete set of tests for all widgets provided by SCADE ARINC 661 Solution.

Configuring and Running Tests

You may configure and run tests from the SCADE installation directory as follows:

1. Create a Server Creator project on your disk.
2. In `<SCADE_INSTALL>\SCADE A661\test\widgets`, customize the `ENVIRONMENT.py` file as follows:
   - Add new variable `CUSTOM_SERVER_PROJECT_DIR` to specify the path to the Server Creator project.
   - Update the following variables `A661_XML_PATH`, `INSTR_SERVER_PATH`, `SERVER_BUILD_PATH`, `SERVER_PATH`, `WIDGET_LIB` to start from `CUSTOM_SERVER_PROJECT_DIR`.
3. In `<SCADE_INSTALL>\SCADE A661\test\widgets`, run `run_test_session.bat`.

All these instructions, as well as added usage details, are available in a `README` file in `<SCADE_INSTALL>\SCADE A661\test\widgets`.

Extending Widget Tests

You may start extending tests for a custom CDS project as follows:

1. Copy the `<SCADE_INSTALL>\SCADE A661\test\widgets` folder to create a working directory.
2. In the working directory, copy `ENVIRONMENT_TEMPLATE.py` into `ENVIRONMENT.py` and customize as follows:
   - Update `SCADE_DIR` as an absolute path.
   - Add new variable `CUSTOM_SERVER_PROJECT_DIR` to specify the path to the Server Creator project.

- Update the following variables `A661_XML_PATH`, `INSTR_SERVER_PATH`, `SERVER_BUILD_PATH`, `SERVER_PATH`, `WIDGET_LIB` to start from `CUSTOM_SERVER_PROJECT_DIR`.

3 In the working directory, customize `run_test_session.bat` as follows:

- Update `SCADE_DIR` with a path to your SCADE installation directory.

4 Copy-paste an existing Python file in `<working_directory>/tests/<your_widget>` folder:

- Name the new Python file with a unique HLR name (*e.g.*, `test_hlr_999.py`).
- Customize the file contents to have one or more test cases with your custom logic. Make sure test case classes and functions are properly renamed with your custom HLR number.
- Run only your new test from the working directory with:
  `run_test_session.bat -w PicturePushButton -k 999`

Appendixes

A / Testing Functions

This appendix provides a systematic description for all the testing functions provided by the testing module.

Initialization

Start Server

TestFramework.**start_server***(server_exe_path: str = 'A661ServerTest.exe', debug: str = '', log_file_path: str = '', delay: int = 3, port: int = 0, udp: bool = False, name: str = '', proxy_ua: int = 0, bg_exec: bool = False, extra: str = '', df_list: Iterable[str] | None = None, conf: str | None = None)* → TestFramework

Launches the execution of the server with its control proxy.

Arguments

server_exe_path: the path on the server binary
debug: not used
log_file_path: the name of output file for server log traces
delay: the period of time after the server is launched
port: the socket port used to communicate with server (by default 1230)
udp: UDP/IP protocol when True, TCP/IP protocol when False (default is False)
name: name displayed in header of DU windows (by default "A661ServerTest")
def_list: list of binary definition files
conf: optional XML server configuration filename
proxy_ua: Server Control Proxy UA ID
bg_exec: enables background execution (no graphical windows)
extra: optional extra parameters to be passed to the server

Returns

The current TestFramework object

Compatibility

From 2024 R2 release onward, the server is started with its definition(s) binary files and configuration files. The definition phase commands expected after "Initialize Connection" are no longer necessary. To maintain compatibility, it is still possible to call `start_server` without arguments `def_list`, `conf`, and `bg_exec`, and use the definition phase commands after "Initialize Connection": "Load Server Configuration" + "Initialize Server" + "Load Definition File" + "Complete DF Loading Phase" (or directly use "Quick Initialization"). In such case, the server is actually started with proper parameters by "Complete DF Loading Phase" command and connection is then established.

Note

This command is based on the *Server* class defined in the a661libstd module (see "Common Classes" in A661 Widgets API appendix). It creates a new Server object and calls the run method on it. The command expects a list of definition binary files. When using DF objects, they first need to be serialized in binary file using the `generate_binary_hex` utility function (see "Utility Functions" in A661 Widgets API appendix).

The server control proxy UA should not be one of the loaded UAs: an error is raised during server connection if it is loaded.

Register DF

TestFramework.**register_df**(*df: Df*) → None

Registers a Df object to the TestFramework instance.

When a Df object is registered, the decoded CDS notifications is linked to corresponding widget or layer. Typically, when a given notification can have different parameter types or number according to the widget type, a Df object should be created and registered to identify the widget type according to its identifier.

Arguments:

df: instance of Df class

Returns

None

Initialize Connection

TestFramework.**init_connection**(*host: str = 'localHost', port: int = 1230, udp: bool = False*) → TestFramework

Sets up the connection with the server, for the Server Control Proxy and for the UAs loaded on the server.

Once the connection is established, the communication with the server is possible, using the commands described in next section.

Arguments

host: server hostname (default is “localHost”)

port: socket base port (default is 1230)

udp: UDP/IP protocol when True, TCP/IP protocol when False (default is False)

Returns

The current TestFramework object

Important

As the server takes some time to start, a delay of several seconds must be respected between the call to “Start Server” and this call, especially when using a batch file (for example the `sleep` function of the `time` module can be used).

Definition Phase Commands (deprecated)

Deprecated definition phase commands to be used only when the "Start Server"command is used without any definition file.

- "Load Server Configuration"
- "Initialize Server"
- "Load Definition File"
- "Complete DF Loading Phase"
- "Quick Initialization"

Load Server Configuration

TestFramework.**load_server_conf**(*configuration_file: str* = '') → None

Loads the server configuration.

Arguments

configuration_file: the name of the XML server configuration file (when not provided, the server uses the default configuration)

Returns

None

Initialize Server

TestFramework.**init_server**(*background_enable: bool* = *False*) → None

Initializes the server and the graphic system.

This command must be called after "Load Server Configuration".

Arguments

background_enable: enable background execution (no graphical windows).

Returns

None

Load Definition File

TestFramework.**load_definition_file**(*def_instance: Union[str | Df]*) → None

Loads a definition file in the server.

Arguments

def_instance: a path to a binary definition file or an instance of Df class.

- When the df is a Df class object:
  - It is registered for the TestFramework instance, in order to be used when decoding CDS notifications (link to corresponding widget or layer is added)
  - A temporary binary file named `proxydf<num>.bin` is created and this is the file loaded by the server.
- When the df is a file, the CDS notifications are decoded without this link, unless a Df instance is registered using "Get ServerControlProxy Object" function

Returns

None

Complete DF Loading Phase

TestFramework.**complete_definition**() → bool

Completes the DF loading phase (end of initialization).

This command must be called after all the DFs are loaded in the server via "Load Definition File".

Arguments

N/A

Returns

A Boolean indicating if pending blocks (corresponding to A661 CDS notifications)

Quick Initialization

TestFramework.**quick_init**(*df: Df | str], configuration_file: str = '', port: int = 1230, udp : bool = False*) → bool

Sets up the connection to the server and executes the definition phase commands (to be used when only one DF is loaded on the server).

This function executes in sequence the following commands: "Initialize Connection" + "Load Server Configuration" + "Initialize Server" + "Load Definition File" + "Complete DF Loading Phase".

Arguments

df: a path to a binary definition file or an instance of Df class

configuration_file: XML server configuration filename (when not provided, the server uses the default configuration)

port: a socket port (default is 1230)

udp: UDP/IP protocol when True, TCP/IP protocol when False (default is False)

Returns

A Boolean indicating if pending blocks (corresponding to A661 CDS notifications)

Main Functions

Once the connection with the server is established, the main functions can be used. These functions use the `ServerControlProxy` class (see "Server Control Proxy" in A661 Widgets API) to send commands to the server and to wait for the corresponding notifications.

Step

TestFramework.**step**(*steps: int = 1, until_has_blocks: bool = False*) → Dict[str, Any]

Runs the server for a given number of steps, or until the server notification is pending.

Arguments

steps: the number of steps the server will run

until_has_blocks: when set to True, runs until the server produces an A661 notification

Returns

A dictionary: *{"nb": <number of steps>, "has_blocks": True|False}*

- *"nb"*: gives the number of executed steps (0 means that an error occurs)
- *"has_blocks"*: indicates that pending blocks (corresponding to A661 CDS notifications) are available

Pull Block

TestFramework.**pull_block**(df: *Df* | *None* = None) → List[Dict[str, Any]]

Gets all pending CDS notifications from the server.

Arguments

df: instance of Df class (optional and only used for widget event). This option is deprecated (df should be registered during initialization using "Get ServerControlProxy Object" function).

Returns

A list of A661 CDS notifications, or an empty list if no notification is available.

Each notification is a dictionary containing at least the following information:

```
{
   "app_id": *application id*,
   "layer_id": *layer id*,
   "context": *context number of the layer*,
   "kind": *notification kind*,
   "notif": *notification content according to kind*
}
```

notification kind is one of the following:

- Widget Event (A661_NOTIFY_WIDGET_EVENT):
  The following keys are also present in the dictionary:
  - "widget_id": widget id
  - "origin": notification origin (always equals to 1 in current implementation)

  "notif" is an event notification object <event>

  When a Df object is provided, <event> object contains links to the corresponding layer and widget objects.
- Layer (A661_NOTIFY_LAYER_EVENT):
  "notif" is a dictionary with "*event*" key giving the event id and "*value*" key giving the event value.
- Exception (A661_NOTIFY_EXCEPTION):
  "notif" is a dictionary with *"event"* key giving the event id and *"value"* key giving a tuple of 3 elements:
  - the reason of the exception (tuple with error code and error name)
  - the widget id at the origin of the exception (when applicable, otherwise 0)
  - the parameter involved in the exception (when applicable tuple with parameter id and name, otherwise (0, NONE))

Push Block

TestFramework.**push_block**(*data: List[Layer | Widget | ExtensionWidget] | Layer| Widget | ExtensionWidget | Msg | bytes | str, layer_id: int | None = None, app_id: int = 1, context_nb: int = 0*) → None

Sends an A661 block to the server.

Arguments

data: the data sent to the server can correspond to:

- a Widget or ExtensionWidget object to build a set parameter command
- a Layer object to build a layer ua request
- a list of Widget, ExtensionWidget and Layer objects
- an Msg object or a bytes type to send a raw block to the server
- a string corresponding to raw value in hexadecimal (*e.g.*, “CA02FA0010” or “CA.02.FA.00.10”)

layer_id: layer identifier (only used if no Layer object in data)

app_id: application identifier

context_nb: context number

Returns

None

When *data* is a Msg object, a bytes or an hexa string, it corresponds to the full raw A661 block, starting with `A661_BEGIN_BLOCK.` or a string representing the data in hexadecimal (with or without ‘.’ separator between each byte). *layer_id* and *context_nb* parameters are ignored in this case (as they are part of the A661 block).

Otherwise, the A661 block is built as follows:

- For a set parameter command (A661_CMD_SET_PARAMETER):

  <widget object>(<widget_id>).<message name>(<message value>)

  The message name is the name of the parameter ident in lower case without A661_ prefix. For example, to build message A661_STYLE_SET to #110 for ToggleButton widget #3:

  ```
  ToggleButton(3).style_set(110)
  ```

- For a layer request:

  Layer(<layer_id>).<message name>(<message value>...)

  The message name is the name of the layer request in lower case without A661_ prefix. For example, to build message A661_REQ_LAYER_ACTIVE for layer #1:

```
Layer(1).req_layer_active()
```

An A661 block is sent to a given layer. It is not possible to provide Layer objects with different ids in *data*. The layer id is provided by the *layer_id* parameter or the id of any Layer object provided in *data*. If there is no Layer object in *data* and *layer_id* is not set, then the A661 block is sent to layer #1.

Push Pointer

TestFramework.**push_pointer**(*x_org: int = -1, y_org: int = -1, button: Union[int, str] = <PointerButton.NONE: 0>, state: Union[int, str] = <PointerState.MOVE: 0>, modifiers: Union[int, str] = <PointerModifier.NONE: 0>, device_id: int = 1, du_id: int = 1, window_id: int = 0, user_unit: bool = False*) → None

Sends a pointer device event to the server.

Arguments

x_org: X origin position of the pointer (-1 means keep same position)

y_org: Y origin position of the pointer (-1 means keep same position)

button: pointer button identifier (enum *Server.PointerButton* or string corresponding to the enum, *e.g.* 'LEFT')

Possible values are: NONE, LEFT, RIGHT, MIDDLE, MAIN_WHEEL_UP, MAIN_WHEEL_DOWN, SECOND_WHEEL_UP, SECOND_WHEEL_DOWN

state: pointer state (enum *Server.PointerState* or string corresponding to the enum, *e.g.* 'PUSHED')

Possible values are: MOVE, PRESSED, RELEASED

modifiers: modifier key identifier (enum *Server.PointerModifiers* or string corresponding to the enum, *e.g.* 'SHIFT')

Possible values are: NONE, SHIFT, CTRL (or a bitfield combination of them)

device_id: pointer device identifier

du_id: DU identifier

window_id: Window index

user_unit: If True, position is expressed in User Units (1/100mm), otherwise it is in Pixels (default is False). Note that the data sent to the server is always in Pixels, therefore the conversion is applied before sending it.

Returns

None

Push Keyboard

TestFramework.**push_keyboard**(*key_code: Union[int, str] = 0, modifiers: Union[int, str] = <KeyboardModifier.NONE: 0>, device_id: int = 1*) → None

Sends a keyboard device event to the server.

Arguments

key_code: keyboard key code (ASCII character, numerical value, enum *Server.KeyboardServerKeys* or string corresponding to the enum, eg 'UP')

Possible values when using the enum are: F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, LEFT, UP, RIGHT, DOWN, PAGE_UP, PAGE_DOWN, HOME, END, BACKSPACE, TAB, LF, CR, ESC, DEL

modifiers: keyboard modifier key identifier (enum *Server.KeyboardModifiers* or string corresponding to the enum, *e.g.* 'SHIFT')

Possible values are: NONE, SHIFT, CTRL, ALT (or a bitfield combination)

device_id: keyboard device identifier

Returns

None

Push Touch

TestFramework.**push_touch**(*x_org: int = 0, y_org: int = 0, state: Union[int, str] = <TouchState.MOVE: 0>, timestamp: int = 0, touch_point_id: int = 1, device_id: int = 1, du_id: int = 1, window_id: int = 0, user_unit: bool = False*) → None

Sends a touch device event to the server.

Arguments

x_org: X origin position of the touch point

y_org: Y origin position of the touch point

state: touch point state (enum *Server.TouchState* or string corresponding to the enum, *e.g.,* 'DOWN')

Possible values are: MOVE, DOWN, UP, LEAVE, STILL, IDLE

timestamp: timestamp in micro-seconds

touch_point_id: touch point identifier

device_id: touch device identifier

du_id: DU identifier

window_id: window index

user_unit: If True, position is expressed in User Units (1/100mm), otherwise it is in Pixels (default is False). Note that the data sent to the server is always in Pixels, therefore the conversion is applied before sending it.

Returns

None

Push Gesture

TestFramework.**push_gesture**(*type_gesture: int | str = GestureType.TAP, number_points: int = 1, state: int | str = GestureState.START, pos_x: int = 0, pos_y: int = 0, bounding_x1: int = 0, bounding_y1: int = 0, bounding_x2: int = 0, bounding_y2: int = 0, add_value1: float = 0.0, add_value2: float = 0.0, device_id: int = 1, du_id: int = 1, user_unit: bool = True*) → None

Sends a gesture device event to the server.

Arguments:

type_gesture: gesture type (enum Server.GestureType or string corresponding to the enum, *e.g.* 'TAP')

Possible values: NONE, TAP, DOUBLETAP, PRESSHOLD, PINCH, ROTATE, DRAG, FLICK, HOLD, TAPCONFIRMED

number_points: number of touch points

state: gesture state (enum *Server.GestureState* or string corresponding to the enum, *e.g.* 'START')

Possible values: MOVE, START, UPDATE, END, ABORT.

For stateless gestures (TAP, DOUBLETAP, TAPCONFIRMED, and FLICK), the value is ignored (always set to NONE)

pos_x: X position of gesture

pos_y: Y position of gesture

bounding_x1: X first position of bounding box (not required if only 1 touch point)

bounding_y1: Y first position of bounding box (not required if only 1 touch point)

bounding_x2: X second position of bounding box (not required if only 1 touch point)

bounding_y2: Y second position of bounding box (not required if only 1 touch point)

add_value1: first additional value depending on gesture type

add_value2: second additional value depending on gesture type

device_id: gesture device identifier

du_id: DU identifier

user_unit: If True, position is expressed in User Units (1/100mm), otherwise it is in Pixels (default is True). Note that the data sent to the server is always in User Units, therefore the conversion is applied before sending it.

Returns

None

Get Parameter

TestFramework.**get_parameter**(*widget_id: int, param_name: str, layer_id: int = 1, app_id: int = 1, first_elem: int = 0, nb_elems: int = 0*) → Any

Gets the type and the value for the widget parameter.

Arguments

widget_id: widget identifier

param_name: widget parameter name

layer_id: layer identifier

app_id: application identifier

first_elem: index of first element to get in case of array parameter

nb_elems: number of elements to get in case of array parameter (0 means all elements)

Returns

The returned parameter value typed according to the parameter type (e.g., the value is an integer if the parameter is an integer).

When the parameter type is complex (pointer, structure), its returned value corresponds to a bytes memory dump.

Get Focused Widget

TestFramework.**get_focused_widget**(*layer_id: int = 1, app_id: int = 1*) → int

Gets the widget ID of the focused widget for the given layer/application.

Arguments

layer_id: layer identifier

app_id: application identifier

Returns

The focused widget ID for the given layer/application, or 0 if no widget has focus.

Get Popup Stack

TestFramework.**get_popup_stack**(*layer_id: int = 1, app_id: int = 1*) → List[int]

Gets the list of widget ID whose popups are currently opened.

Arguments

layer_id: layer identifier

app_id: application identifier

Returns

The list of widget ID whose popups are currently opened for the given layer/application, or empty list if no popup is opened.

The order in the list is the display order (the first one is displayed first, so the last one is the one on top of the others)

Get Widgets under Cursor

TestFramework.**get_widgets_under_cursor**(*layer_id: int = 1, app_id: int = 1, full: bool = False*) → List[int]

Gets the list of widget ID having the cursor.

Arguments

layer_id: layer identifier

app_id: application identifier

full: when set to False, only returns the ID of the widgets under the cursor. When set to True, returns the full report of all widgets, including area IDs/item IDs.

Returns

List of widgets having the cursor for the given layer/application, or an empty list if no widget has the cursor. The order in the list is the display order (the first one is displayed first, so the last one is the one on top of the others).

If full = False (default): for each widget, only its ID is given

If full = True: for each widget, a dictionary is returned with following keys:

- 'app_id': application id given as parameter
- 'layer_id': layer id given as parameter
- 'widget_id': widget ID
- 'is_mapitem': True if the widget is a Map(Horz/Vert)_ItemList
- 'area_ids': list of areas IDs under the cursor for the widget, or list of item indexes in case of mapitem widget

Get Widgets under Gesture

TestFramework.**get_widgets_under_gesture**(*layer_id: int = 1, app_id: int = 1, type_gesture: int | str = GestureType.TAP, full: bool = False*) → List[int]

Gets the list of widget ID having a given type of gesture.

Arguments

layer_id: layer identifier

app_id: application identifier

type_gesture: gesture type (enum *Server.GestureType* or string corresponding to the enum, e.g., 'TAP')

Possible values: TAP, DOUBLETAP, PRESSHOLD, PINCH, ROTATE, DRAG, FLICK, HOLD, TAPCONFIRMED

full: when set to False, only returns the ID of the widgets under the gesture. When set to True, returns the full report of all widgets, including area IDs/item IDs.

Returns

List of widgets having the gesture for the given layer/application, or an empty list if no widget has the gesture. The order in the list is the display order (the first one is displayed first, so the last one is the one on top of the others).

If full = False (default): for each widget, only its ID is given

If full = True: for each widget, a dictionary is returned with following keys:

- 'app_id': application id given as parameter
- 'layer_id': layer id given as parameter
- 'widget_id': widget ID
- 'is_mapitem': True if the widget is a Map(Horz/Vert)_ItemList
- 'area_ids': list of areas IDs under the gesture for the widget, or list of item indexes in case of mapitem widget

Get Cursor Pointer Data

TestFramework.**get_cursor_pointer_data**(*pointer_id: int, user_unit: bool = False*) → Dict[str, Any]

Gets data of cursor pointer for the given pointer identifier.

Arguments

pointer_id: pointer identifier

user_unit: If True, position is given in User Units (1/100mm), else in Pixels (default is False). Note that the value sent by the server is always in Pixels, therefore the conversion is applied after receiving it.

Returns

A dictionary of cursor pointer data, containing the du identifier, window identifier, positions (in User Unit or Pixels), shape and active of cursor.

{“du_id”: <du_id>, “window_id”: <window_id>, “position_x”: <position_x>, “position_y”: <position_y>, “shape_id”: <shape_id>, “active”: <true|false>}

Get Screenshot

TestFramework.**get_screenshot**(*x_org: int = 0, y_org: int = 0, width: int = 0, height: int = 0, resulting_file_path: Union[str, pathlib.Path] = 'screenshot', du_id: int = 1, user_unit: bool = False*) → Dict

Gets a screenshot of the Display Unit.

Arguments

x_org: X origin position

y_org: Y origin position

width: width size of screenshot

height: height size of screenshot

resulting_file_path: a string or a pathlib.Path object

In case of string, it gives the path to the saved screenshot file, to which the du ID, timestamp, and .png extensions are added.

In case of Path object, the object provides the screenshot full filename.

du_id: DU identifier

user_unit: If True, the requested position is expressed in User Units (1/100mm), otherwise it is in Pixels (default is False). Note that the data sent to the server is always in Pixels, therefore the conversion is applied before sending it.

Returns

A dictionary containing the actual position, the size of the screenshot, and the path of the screenshot file

{“origin_x”: <origin_x>, “origin_y”: <origin_y>, “width”: <width>, “height”: <height>, “file”:<png_file>}

Position and size are expressed in Pixels.

The RGBA buffer is saved into the PNG file <png_file>.

If <resulting_file_path> is a string, the filename is built as follows: *<resulting_file_path>-du<du_id>-<timestamp>.png*. Otherwise, the Path object provides the filename.

The size of screenshot is given by <width> and <height> if value is greater than 0, otherwise the width or height of the Display Unit is taken.

If the rectangle area of the requested screenshot is not fully included in the display unit area, it is resized accordingly.

If its position is outside the display unit area, it is reset to the origin of the display unit.

Get Interface Data

TestFramework.**get_interface_data**(*interface_name: str, window_index: int = 0, app_id: int = 0, layer_id: int = 0, widget_id: int = 0*) → Any

Gets the data of a given interface.

Supported interfaces are:

- Focus (data at window and widget levels)
- Interactivity (data at window and widget levels)
- SensitiveArea (data at window level)
- TabbedPanel (data at widget level)
- Touch (data at server and window levels)

Arguments

interface_name: interface name

window_index: display window index (when applicable)

app_id: application identifier (optional)

layer_id: layer identifier (required only if app_id is provided)

widget_id: widget identifier (required only if app_id is provided)

Returns

A dictionary containing the parameters of the interface.

When the widget identifier is provided, the key “widget_interface” contains a sub dictionary for the interface parameters related to the widget, when it is applicable.

Set Cursor Active State

TestFramework.**set_cursor_active_state**(*cursor: int = 0, active: bool = True*) → List[bool]

Sets cursor pointer active state.

Arguments

cursor: cursor number (from 1 to the number of cursor pointers declared in the server conf). 0 means no change (used to get the list of active cursors)

Returns

A list of boolean corresponding to the active state of each cursor

Set Interface Value

TestFramework.**get_interface_value**(*widget_id: int, interface_name: str, method_name: str, value: int, layer_id: int = 1, app_id: int = 1*) → None

Forces the value of a given method (interface input) for a server or widget interface.

Supported interfaces/methods are:

- ForceEvent/ForceEvent (value=message ID)
- Keyboard/PredefKeys (value=ulong range)
- PopUpMenu/CloseRequest (value=0/1)
- Selection/Select (value=0/1)
- UAState/NoService (value=0/1)

Arguments

widget_id: widget identifier

interface_name: interface name

method_name: method name for the given interface

value: value to set (integer or string corresponding to method enum or constant)

layer_id: layer identifier

app_id: application identifier

Returns

None

Run Interactive Session

TestFramework.**run_interactive_session**(*number_of_steps: int = 500, stop_condition: int = <StopCondition.NONE: 0>, output_sequence_path: str = 'record', save_scn: bool = False, notif_kind: int = 0, notif_id: int = 0, app_id: int = 0, layer_id: int = 0, widget_id: int = 0*) → List[Dict]

Runs the server in interactive mode (*i.e.,* with its input devices enabled) for a given number of steps, or until a stop condition is met.

The command asks the server to run in standard interactive mode (physical devices are responsive) for a given number of steps (-1 means unlimited number of steps), or when *stop_condition* is met.

The *stop_condition* gives the list of input devices or A661 notification (logical "OR" of POINTER, KEYB, TOUCH, GESTURE and NOTIF values, or NONE when no stop condition is requested) for which the presence of an event stops the interactive execution.

When stop condition NOTIF is selected, the execution stops as soon as a given notification identified by its kind, id, app id, layer id and widget id is present (a 0 value means any value, so to stop on the presence of any notification, all values shall be set to 0).

Arguments

number_of_steps: a number of steps to execute

stop_condition: a condition to end the interactive session (enum *StopCondition*)

output_sequence_path: path to the JSON file where to serialize the returned data

notif_kind: kind of CDS notification (only used when stop condition NOTIF is selected)

notif_id: notification identifier (only used when stop condition NOTIF is selected)

app_id: application identifier (only used when stop condition NOTIF is selected)

layer_id: layer identifier (only used when stop condition NOTIF is selected)

widget_id: widget identifier (only used when stop condition NOTIF is selected)

Returns

The list of device events and CDS notifications, with, for each element, a dictionary containing *{"step": <step_number>, "device_events": <device_events> or "cds_notifications": <notifications>}* (see details below)

When <*output_sequence_path*> is different from empty string, the returned data is serialized in JSON file <output_sequence_path>.json.

<device_events> is a list of device events (POINTER, KEYB, TOUCH or GESTURE):

```
- POINTER:
    {
        "kind": "POINTER",
        "dev_id": <device_id>,
        "du_id": <du_id>,
        "dw_id": <window_id>,
        "button": <button>,
        "state": <state>,
        "modifiers": <modifiers>,
        "x": <x>,
        "y": <y>
    }
- KEYB:
    {
        "kind": "KEYB",
        "dev_id": <device_id>,
        "keycode": <keycode>,
        "modifiers": <modifiers>
    }
- TOUCH:
    {
        "kind": "TOUCH",
        "dev_id": <device_id>,
        "du_id": <du_id>,
        "dw_id":<window_id>,
        "touch_point_id": <touch_point>,
        "state": <state>,
        "x": <x>,
        "y": <y>,
        "timestamp": <timestamp>
    }
- GESTURE:
    {
        "kind": "GESTURE",
        "dev_id": <device_id>,
        "type": <type>,
        "numberOfPoints": <numberOfPoints>,
        "state": <state>,
        "posX": <posX>,
        "posY": <posY>,
        "boundingX1": <boundingX1>,
        "boundingY1": <boundingY1>,
        "boundingX2": <boundingX2>,
        "boundingY2": <boundingY2>,
        "addValue1": <addValue1>,
        "addValue2": <addValue2>
    }
```

<notifications> is the list of CDS notification (exception, layer events, and widget events). See content details in “Pull Block” command.

Change Active Cursor Pointer

TestFramework.**change_active_cursor**(*cursor: int = 0*) → Dict[str, Any]

Changes active cursor pointer and run the server for one step.

Arguments

cursor: New active cursor number (from 1 to the number of cursor pointers declared in the server conf). 0 means select next cursor (same as CTRL+F2 on on server when input devices are enabled).

Returns

A dictionary: *{"cursor": <new cursor number>, "has_blocks": True|False}*

- "*cursor*": gives the new active cursor number (0 means that an error occurs)
- "*has_blocks*": indicates that pending blocks (corresponding to A661 CDS notifications) are available

Change Display Conf

TestFramework.**change_display_conf**(*conf: int = 0*) → Dict[str, Any]

Changes display configuration and run the server for one step.

Arguments

conf: New display configuration number (from 1 to the number of display configurations). 0 means select next conf (same as CTRL+F1 on on server when input devices are enabled).

Returns

A dictionary: *{"conf": <new conf number>, "has_blocks": True|False}*

- "*conf*": gives the new display configuration number (0 means that an error occurs)
- "*has_blocks*": indicates that pending blocks (corresponding to A661 CDS notifications) are available

Check Functions

The `a661testcheck.py` module provides functions to check the widget observed data (A661 notifications, widget parameter, or screenshot).

Check Layer Event

check_layer_event*(event: str, notifications: List[Dict[str, Any]], expected_values: Optional[Any] = None, app_id: int = 1, layer_id: int = 1, notpresent: bool = False)* → bool

Checks presence and content of a given layer event from a list of notifications.

The list of notifications is by default retrieved as return of "Pull Block" or "Run Interactive Session" commands.

Arguments

event: the layer event identifier name (e.g. "A661_NOTE_LAYER_IS_ACTIVE")

notifications: List of notifications.

expected_values: the expected values of the layer event parameters (if any):

- When the layer event has more than one parameter, use a tuple to provide the different parameter values (None can be used to ignore one of the parameters)
- The check can be done on a set of possible values. In this case, the values are provided in a list (list of tuples in case of several parameters)
- Examples:
  - one parameter (*e.g.,* int): *1* if only "1" is expected, *[1, 2, 5]* if one of these values is expected
  - several parameters (*e.g.,* int, array): *(3, [1,2,3])* or *(2, None)* or *[(None,[1,2]),(3,[2,6,8])]*

app_id: the application identifier of expected event (if different from 1)

layer_id: the layer identifier of expected event (if different from 1)

notpresent: when set to True, checks that the layer event is not present in the list of notifications

Returns:

A Boolean value, which is True when:

- the layer event with corresponding <*expected_values*> (or any of the values if <*expected_values*> is a list) is found in the list of notifications returned by last command or provided in <*notifications*>. If <*expected_values*> is equal to None, then no check is done on the parameters value.
- the layer event is not found in the list of notifications when <*notpresent*>=*True* (in this case <*expected_values*> is ignored)

Otherwise, the returned value is False.

Check Widget Event

check_widget_event(*event: str, widget_id: int, notifications: List[Dict[str, Any]], expected_values: Optional[Any] = None, app_id: int = 1, layer_id: int = 1, notpresent: bool = False*) → bool

Checks presence and content of a given widget event from a list of notifications.

The list of notifications is by default retrieved as return of "Pull Block" or "Run Interactive Session" commands.

Arguments

event: the widget event identifier name (e.g. "A661_EVT_STATE_CHANGE")

widget_id: the widget identifier of expected event

notifications: list of notifications.

expected_values: the expected values of the widget event parameters (if any):

- When the widget event has more than one parameter, use a tuple to provide the different parameter values (None can be used to ignore one of the parameters)
- The check can be done on a set of possible values. In this case, the values are provided in a list (list of tuples in case of several parameters)
- Examples:
  - one parameter (*e.g.,* int): *1* if only "1" is expected, *[1, 2, 5]* if one of these values is expected
  - several parameters (*e.g.,* int, array): *(3, [1,2,3])* or *(2, None)* or *[(None,[1,2]),(3,[2,6,8])]*

app_id: the application identifier of expected event (if different from 1)

layer_id: the layer identifier of expected event (if different from 1)

notpresent: when set to True, check that the widget event is not present in the list of notifications

Returns

A Boolean value, which is True when:

- the widget event with corresponding <*expected_values*> (or any of the values if <*expected_values*> is a list) is found in the list of notifications returned by last command or provided in <*notifications*>. If <*expected_values*> is equal to None, then no check is done on the parameters value.
- the widget event is not found in the list of notifications when <*notpresent*>=*True* (in this case <*expected_values*> is ignored)

Otherwise, the returned value is False.

Check Exception

check_exception(*event: str, notifications: List[Dict[str, Any]], reason: str = '', widget_id: Union[int, List[int]] = - 1, parameter: Union[str, List[str]] = '', app_id: int = 1, layer_id: int = 1, notpresent: bool = False*) → bool

Checks presence and content of a given exception from a list of notifications.

The list of notifications is by default retrieved as return of "Pull Block" or "Run Interactive Session" commands.

Arguments:

event: the exception identifier name (e.g., "A661_ERR_SET_ABORTED").

notifications: list of notifications.

reason: the expected reason of the exception (error code or error name).

- If "" is provided (empty string), then no check is done on the reason.
- Check can be done on a set of possible values. In this case, values are provided in a list.

widget_id: the expected widget identifier associated with the exception.

- If -1 is provided, then no check is done on the widget identifier.
- Check can be done on a set of possible values. In this case, values are provided in a list.

parameter: the expected widget parameter (identifier or name) associated with the exception.

- If "" is provided (empty string), then no check is done on the parameter name.
- Check can be done on a set of possible values. In this case, values are provided in a list.

app_id: the application identifier of expected exception (if different from 1)

layer_id: the layer identifier of expected exception (if different from 1)

notpresent: when set to True, checks that the exception is not present in the list of notifications

Returns

A Boolean value, which is True when:

- the exception with corresponding <*reason*>, <*widget_id*> and <*parameter*> (or any of the list elements if lists are used) is found in the list of exceptions returned by last command or provided in <*notifications*>.
- the exception is not found in the list of exceptions when <*notpresent*>=*True*

Otherwise, the returned value is False.

Check Parameter

check_parameter(*expected_values: Any, param_value: Any]*) → bool

Checks widget parameter value after a call to "Get Parameter" command.

Arguments

expected_values: the expected values for checking the widget parameter value.

- Check can be done on a set of possible values. In this case, values are provided in a list.

param_value: value corresponding to return of "Get Parameter" command.

Returns

A Boolean value, which is True when <*expected_values*> (or any of the values if <*expected_values*> is a list) corresponds to the value returned by last command or provided in <*returned_value*>.

Otherwise, the returned value is False.

Check Image

check_image(*actual_file: str, expected_file: str, save_to_file: Union[bool, str] = False, imgdiff_dir: str = ''*) → bool

Checks a screenshot image returned after a call to "Get Screenshot" command with respect to a given expected image, by using of the SDYIMGDIFF tool in SCADE bin directory.

Arguments

actual_file: the path to the actual image file

expected_file: the path to the given expected image

save_to_file: Either a Boolean value or a string

- True means the result of comparison of two images is saved in *comp_screenshots.bmp* file
- False means the result of comparison of two images is not saved
- if a file pathname is given instead of a Boolean, the comparison result is stored in this file

imgdiff_dir: the path on "SDYIMGDIFF.exe" image comparison tool directory. When not set, "..\..\..\SCADE\bin" is used.

Returns

A Boolean whose value is True if the actual image and the given expected image are identical.

Logging Functions

Each module of the 'taf' package uses a logger whose name is the name of the module. The 'taf' logger has no handler defined, so the handlers of the root logger are used by default. It is the responsibility of end users to properly configure the 'taf' logger or the root logger so that the logs are properly displayed. The `a661log.py` module provides two functions to be applied on the 'taf' logger, to choose to log on the console and/or in a file. Here is an example of what can be done, and might be updated or copied for more specific needs.

Log Console

log_console(*on: bool = True, level: Optional[Union[int, str]] = None, infofilter: Optional[List[str]] = None*) → None

Logs traces on console.

Arguments

on: Set to True to display the traces on the console

level: logging level for the console handler (only effective if its value is higher than the logger level, as expected by logging behavior)

If the logger level is not set (level = 0), then for handler level less than WARNING (30), the logger level will be set to the handler level (otherwise, the log will not be displayed)

infofilter: log info level filters (list of strings), to filter the info level logs.

When set and not empty, any message not containing any of the strings of the list will be filtered.

For example, to have only the "CHECK" logs, add "CHECK" as filter string.

An empty list removes all the filters, therefore no log is filtered.

Returns

None

Log File

log_file(*on: bool = True, log_file_path: str = 'test_trace.log', level: Optional[Union[int, str]] = None, infofilter: Optional[List[str]] = None, reset: bool = False*) → None

Logs traces in file.

Arguments

on: set to True to log traces in file

log_file_path: pathname on log file (if not provided, traces are logged in file 'test_trace.log')

level: logging level for the file handler (only effective if its value is higher than the logger level, as expected by logging behavior)

If the logger level is not set (level = 0), then for handler level less than WARNING, the logger level is set to the handler level (otherwise, the log will not be dumped)

infofilter: log info level filters (list of strings), to filter the info level logs.

When set and not empty, any message not containing any of the strings of the list is filtered.

For example, to have only the "CHECK" logs, add "CHECK" as filter string.

An empty list removes all the filters, therefore no log is filtered.

Returns

None

Example 1: log INFO level traces (and above) on console:

```
# log INFO on console (if the logger level is not set it will be set to INFO
also)
log_console(True, "INFO")
```

Example 2: log all traces on file and only the CHECK info on console:

```
# The logger level has to be set to the lowest level
logging.getLogger("taf").setLevel("DEBUG")
# log DEBUG on file: no need to set the level (will use the logger one)
log_file(True, "mylogs.txt")
# log INFO on console only for CHECK info logs
log_console(True, "INFO", ["CHECK"])
...
```

Utility Functions

Stop Server

TestFramework.**stop_server**(*delay: int* = *5*) → taf.a661test.TestFramework

Shuts down the server started with the `start_server` command.

Arguments

delay: delay in seconds before forcing the server to close (-1 means close is not forced)

Returns

The current TestFramework object

Create DF

static TestFramework.**create_df**(*widgets: Widget | List[Widget] | Dict[int, List[Widget]], ua_id: int = 1, layer_id: int = 1*) → Df

Creates a Df object for one or several layers.

Arguments

widgets:

- a widget object or a list of widget objects (single layer Df, and in this case *layer_id* gives the corresponding layer id).
- a list of dictionaries with one dictionary per layer. Each dictionary contains one element where key is the layer id and value is the list of widget objects (in this case *layer_id* is not used).

ua_id: UA identifier

layer_id: layer identifier (only used for single layer Df when a widget or a list of widgets is provided)

Returns:

a Df object (class Df)

A widget object is defined as follows:

```
<Widget class>(<widget_id>).define(<param>=<value>, ...)
```

The parameter name and value are set according to the A661 standard, and the default value (as defined in the XML description file of the widget) is used when the parameter is not provided to the define() method.

To create a Df with a single widget (by default for layer #1 of UA #1):

```
df = create_df(<widget object>)
```

To create a Df with several widgets inside one layer (by default for layer #1 of UA #1):

```
df = create_df([<widget object>, ...])
```

To create a Df with several layers (by default for UA #1):

```
df = create_df({<layer_id>: [<widget object>, ...], ...})
```

Example:

To create a DF containing a GpArcCircle widget #3 with specific value for PosX, PosY, and Styleset:

```
mydf = create_df(GpArcCircle(3).define(PosX=1000, PosY=1000, StyleSet=110))
```

Note

To be loaded on the server, the created Df object must be serialized as a DF binary file using the `generate_binary_hex` function (see "Utility Functions" in A661 Widgets API). The created Df object must be registered using "Get ServerControlProxy Object" command to link widget and layer objects defined in the Df to decoded layer and widget notifications.

Get ServerControlProxy Object

TestFramework.**get_server_control_proxy**() → ServerControlProxy

Returns the instanciated ServerControlProxy object

Arguments:

None

Returns:

ServerControlProxy object (see "Server Control Proxy" in A661 Widgets API)

B / Coverage Functions

This appendix provides a systematic description for the coverage functions provided by the testing module.

Coverage Functions

The following coverage functions are available:

Coverage Reset

TestFramework.**coverage_reset**() → None

Resets coverage in the instrumented server.

Arguments

N/A

Returns

None

Coverage Dump

TestFramework.**coverage_dump**(*pathname: Optional[str] = None*) → None

Dumps coverage in the instrumented server.

Arguments

pathname: file path without extension, used as basename for dump of raw coverage files (.info and .sdyinfo). If not provided, the current date-time is used as basename.

Returns

None

Example 1: coverage dump at end of session:

```
te = TestFramework()
# Initialize server and Load definition files
...
# Perform tests
...
# Dump coverage before exiting
te.coverage_dump()
te.stop_server()
```

Example 2: coverage dump after load definition:

```
te = TestFramework()
# Initialize server and Load definition files
...
# Dump coverage after Load definition files
te.coverage_dump()
# Etc
...
```

Example 3: several coverage reset and dump:

```
te = TestFramework()
# Initialize server and Load definition files
...
# Perform some commands not needing coverage
...
# Perform first test sequence between reset and dump
te.coverage_reset()
...
te.coverage_dump()
# Perform some commands not needing coverage
...
# Perform second test sequence between reset and dump
te.coverage_reset()
...
te.coverage_dump()
# Etc
...
```

Coverage Report Script

A coverage report script `a661_mc_report.py` is provided to help generate SCADE Test reports from raw coverage produced by coverage_dump.

`a661_mc_report.py`: Generate SCADE Test coverage reports for A661 Widgets.

This script creates a sub-directory SSMC with:

- a SCADE Suite project '`WidgetsSuite.etp`' referencing all Suite widgets models
- a SCADE Test project '`TestSuite.etp`' referencing Suite coverage results and justifications
- the merge of Suite Model Coverage raw results '`*.info`' into '`behavior.info`'
- import in '`TestSuite.etp`' of merged coverage results, using traceability information
- a top-level SCADE Test workspace '`TestSuite.vsw`' referencing '`TestSuite.etp`' and '`WidgetsSuite.etp`'

This script creates a sub-directory 'SDYMC' with:

- a SCADE Display project '`WidgetsDisplay.etp`' referencing all Display widgets models
- a SCADE Test project '`TestDisplay.etp`' referencing Display coverage results and justifications
- the merge of Display Model Coverage raw results '`*.sdyinfo`' into '`display.sdyinfo`'
- import in '`TestDisplay.etp`' of merged coverage results, using traceability information
- a top-level SCADE Test workspace '`TestDisplay.vsw`' referencing '`TestDisplay.etp`' and '`WidgetsDisplay.etp`'

Usage:

```
a661_mc_report.py [-h] [-o OUT] [-r RES] [-j JUSTIF] [-g GEN] [-w WIDGETLIB] [-s SCADE]
```

optional arguments:

-h, --**help**

show this help message and exit

-o OUT, **--out** *OUT*

output directory (will contain SSMC, SDYMC)

-**r RES**, **--res** *RES*

results directory (containing *.info, *.sdyinfo)

-j JUSTIF, **--justif** *JUSTIF*

justifications directory (containing *.mcj, *.sdymcj)

-g GEN, **--gen** *GEN*

generation directory (containing suitegen, displaygen, wwgen)

-w WIDGETLIB, **--widgetlib** *WIDGETLIB*

widgetlib directory (containing widgets, symbolCommands)

-s SCADE, **--scade** *SCADE*

SCADE installation directory

Example: use SCADE Widgetlib, generated code instrumented for Model Coverage, and dumps of coverage raw results:

```
python "C:\Program Files\ANSYS Inc\v<VersionNumber>\SCADE\SCADE
A661\PythonLib\taf\a661_mc_report.py"
        --widgetlib "C:\Program Files\ANSYS Inc\v<VersionNumber>\SCADE\SCADE A661\Server"
        --gen "C:\Program Files\ANSYS Inc\v<VersionNumber>\SCADE\SCADE A661\Server For
Test_output\src"
        --res "C:\Temp\CoverageResults"
        --out "C:\Temp\CoverageReports"
```

C / A661 Widgets API

This appendix provides a systematic description for all the classes and methods of the A661 Widgets API.

Widgets

All widget classes inherit from class **Widget** (see "Widget Class").

- "ActiveArea"
- "AnimationGroup"
- "AnimationOnParam"
- "AnimationRotation"
- "AnimationScale"
- "AnimationTranslation"
- "BasicContainer"
- "BlinkingContainer"
- "BroadcastReceiver"
- "BufferFormat"
- "CheckButton"
- "ComboBox"
- "ComboBoxEdit"
- "Connector"
- "CursorOver"
- "CursorPosOverlay"
- "CursorRef"
- "DataConnector"
- "DataScalingULong"
- "DataScalingFR180"
- "DataScalingLong"
- "EditBoxMasked"
- "EditBoxMultiLine"
- "EditBoxNumeric"
- "EditBoxNumericBCD"
- "EditBoxText"
- "EventHandler"
- "ExternalSource"
- "FocusIn"
- "FocusLink"
- "FocusOut"
- "GestureArea"
- "GpArcCircle"
- "GpArcEllipse"
- "GpCrown"
- "GpLine"
- "GpLinePolar"
- "GpPolyline"
- "GpRectangle"
- "GpTriangle"
- "InkArea"
- "KeyboardArea"
- "Label"
- "LabelComplex"
- "MapBoundary"
- "MapGrid"
- "MapHorz"
- "MapHorz_Container"
- "MapHorz_ItemList"
- "MapHorz_Panel"
- "MapHorz_Source"
- "MapHorz_VertexBuffer"
- "MapVert"
- "MapVert_Container"
- "MapVert_ItemList"
- "MapVert_Panel"
- "MapVert_Source"
- "MaskContainer"
- "MenuBar"
- "MultiStateButton"
- "MutuallyExclusiveContainer"
- "NoServiceMonitor"
- "NumericReadout"
- "PagingContainer"
- "Panel"
- "Picture"
- "PictureAnimated"
- "PicturePushButton"
- "PictureToggleButton"
- "PopUpMenu"
- "PopUpMenuButton"
- "ProxyButton"
- "PushButton"
- "RadioBox"
- "RotationContainer"
- "ScrollList"
- "ScrollPanel"
- "ScrollWheelArea"
- "SelectionListButton"
- "ShuffleToFitContainer"
- "SizeToFitContainer"
- "Slider"
- "Symbol"
- "SymbolAnimated"
- "SymbolPushButton"
- "SymbolToggleButton"
- "TabbedPanel"
- "TabbedPanelGroup"
- "ToggleButton"
- "TouchArea"
- "TranslationContainer"
- "WatchdogContainer"

ActiveArea

class **ActiveArea**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=2000, SizeY=600, StyleSet=1, NextFocusedWidget=0, AutomaticFocusMotion=False*) → ActiveArea

Method to set the parameter(s) of ActiveArea.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for ActiveArea.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for ActiveArea.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for ActiveArea.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for ActiveArea.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for ActiveArea.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for ActiveArea.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for ActiveArea.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for ActiveArea.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for ActiveArea.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for ActiveArea.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for ActiveArea.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for ActiveArea.

AnimationGroup

class **AnimationGroup**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, Type='A661_PARALLEL', Delay=0, AnimationRepetition=0*) → AnimationGroup

Method to set the parameter(s) of AnimationGroup.

animation_delay(*v_delay*) → Widget

Method to build runtime message A661_ANIMATION_DELAY for AnimationGroup.

animation_state(*v_animation_state: int | str | a661_animation_state*) → Widget

Method to build runtime message A661_ANIMATION_STATE for AnimationGroup.

animation_repetition(*v_animation_repetition*) → Widget

Method to build runtime message A661_ANIMATION_REPETITION for AnimationGroup.

AnimationOnParam

class **AnimationOnParam**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, StyleSet=0, TargetWidgetID=0, TargetParamID='A661_SIZE_X', FromValue=1000, ToValue=3000, AnimationLawRef=0, Duration=3000, Delay=0, AnimationRepetition=0*) → AnimationOnParam

Method to set the parameter(s) of AnimationOnParam.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for AnimationOnParam.

from_value(*v_from_value*) → Widget

Method to build runtime message A661_FROM_VALUE for AnimationOnParam.

to_value(*v_to_value*) → Widget

Method to build runtime message A661_TO_VALUE for AnimationOnParam.

animation_duration(*v_duration*) → Widget

Method to build runtime message A661_ANIMATION_DURATION for AnimationOnParam.

animation_delay(*v_delay*) → Widget

Method to build runtime message A661_ANIMATION_DELAY for AnimationOnParam.

animation_state(*v_animation_state: int | str | a661_animation_state*) → Widget

Method to build runtime message A661_ANIMATION_STATE for AnimationOnParam.

animation_repetition(*v_animation_repetition*) → Widget

Method to build runtime message A661_ANIMATION_REPETITION for AnimationOnParam.

animation_law_ref(*v_animation_law_ref*) → Widget

Method to build runtime message A661_ANIMATION_LAW_REF for AnimationOnParam.

AnimationRotation

class **AnimationRotation**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, TargetWidgetID=0, StyleSet=0, CenterX=0, CenterY=0, FromAngle=0.0, ToAngle=90.0, AnimationLawRef=0, Duration=3000, Delay=0, Direction='A661_CLOCKWISE', NumberOfTurns=0, AnimationRepetition=0*) → AnimationRotation

Method to set the parameter(s) of AnimationRotation.

from_angle(*v_from_angle*) → Widget

Method to build runtime message A661_FROM_ANGLE for AnimationRotation.

to_angle(*v_to_angle*) → Widget

Method to build runtime message A661_TO_ANGLE for AnimationRotation.

center_x(*v_center_x*) → Widget

Method to build runtime message A661_CENTER_X for AnimationRotation.

center_y(*v_center_y*) → Widget

Method to build runtime message A661_CENTER_Y for AnimationRotation.

center_xy(*v_center_x*, v_center_y) → Widget

Method to build runtime message A661_CENTER_XY for AnimationRotation.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for AnimationRotation.

animation_duration(*v_duration*) → Widget

Method to build runtime message A661_ANIMATION_DURATION for AnimationRotation.

animation_delay(*v_delay*) → Widget

Method to build runtime message A661_ANIMATION_DELAY for AnimationRotation.

animation_state(*v_animation_state*: *int | str | a661_animation_state*) → Widget

Method to build runtime message A661_ANIMATION_STATE for AnimationRotation.

animation_repetition(*v_animation_repetition*) → Widget

Method to build runtime message A661_ANIMATION_REPETITION for AnimationRotation.

animation_law_ref(*v_animation_law_ref*) → Widget

Method to build runtime message A661_ANIMATION_LAW_REF for AnimationRotation.

direction(*v_direction: int | str | a661_rotation_direction*) → Widget

Method to build runtime message A661_DIRECTION for AnimationRotation.

number_of_turns(*v_number_of_turns*) → Widget

Method to build runtime message A661_NUMBER_OF_TURNS for AnimationRotation.

AnimationScale

class **AnimationScale**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, TargetWidgetID=0, StyleSet=0, FromScaleX=1.0, FromScaleY=1.0, ToScaleX=2.0, ToScaleY=2.0, AnimationLawRef=0, Duration=3000, Delay=0, AnimationRepetition=0*) → AnimationScale

Method to set the parameter(s) of AnimationScale.

from_scale_x(*v_from_scale_x*) → Widget

Method to build runtime message A661_FROM_SCALE_X for AnimationScale.

from_scale_y(*v_from_scale_y*) → Widget

Method to build runtime message A661_FROM_SCALE_Y for AnimationScale.

from_scale_xy(*v_from_scale_x, v_from_scale_y*) → Widget

Method to build runtime message A661_FROM_SCALE_XY for AnimationScale.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for AnimationScale.

to_scale_x(*v_to_scale_x*) → Widget

Method to build runtime message A661_TO_SCALE_X for AnimationScale.

to_scale_y(*v_to_scale_y*) → Widget

Method to build runtime message A661_TO_SCALE_Y for AnimationScale.

to_scale_xy(*v_to_scale_x, v_to_scale_y*) → Widget

Method to build runtime message A661_TO_SCALE_XY for AnimationScale.

animation_duration(*v_duration*) → Widget

Method to build runtime message A661_ANIMATION_DURATION for AnimationScale.

animation_delay(*v_delay*) → Widget

Method to build runtime message A661_ANIMATION_DELAY for AnimationScale.

animation_state(*v_animation_state: int | str | a661_animation_state*) → Widget

Method to build runtime message A661_ANIMATION_STATE for AnimationScale.

animation_repetition(*v_animation_repetition*) → Widget

Method to build runtime message A661_ANIMATION_REPETITION for AnimationScale.

animation_law_ref(*v_animation_law_ref*) → Widget

Method to build runtime message A661_ANIMATION_LAW_REF for AnimationScale.

AnimationTranslation

class **AnimationTranslation**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, TargetWidgetID=0, StyleSet=0, FromValueX=0, FromValueY=0, ToValueX=3000, ToValueY=3000, AnimationLawRef=0, Duration=3000, Delay=0, AnimationRepetition=0*) → AnimationTranslation

Method to set the parameter(s) of AnimationTranslation.

from_value_x(*v_from_value_x*) → Widget

Method to build runtime message A661_FROM_VALUE_X for AnimationTranslation.

from_value_y(*v_from_value_y*) → Widget

Method to build runtime message A661_FROM_VALUE_Y for AnimationTranslation.

from_value_xy(*v_from_value_x, v_from_value_y*) → Widget

Method to build runtime message A661_FROM_VALUE_XY for AnimationTranslation.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for AnimationTranslation.

to_value_x(*v_to_value_x*) → Widget

Method to build runtime message A661_TO_VALUE_X for AnimationTranslation.

to_value_y(*v_to_value_y*) → Widget

Method to build runtime message A661_TO_VALUE_Y for AnimationTranslation.

to_value_xy(*v_to_value_x, v_to_value_y*) → Widget

Method to build runtime message A661_TO_VALUE_XY for AnimationTranslation.

animation_duration(*v_duration*) → Widget

Method to build runtime message A661_ANIMATION_DURATION for AnimationTranslation.

animation_delay(*v_delay*) → Widget

Method to build runtime message A661_ANIMATION_DELAY for AnimationTranslation.

animation_state(*v_animation_state: int | str | a661_animation_state*) → Widget

Method to build runtime message A661_ANIMATION_STATE for AnimationTranslation.

animation_repetition(*v_animation_repetition*) → Widget

Method to build runtime message A661_ANIMATION_REPETITION for AnimationTranslation.

animation_law_ref(*v_animation_law_ref*) → Widget

Method to build runtime message A661_ANIMATION_LAW_REF for AnimationTranslation.

BasicContainer

class **BasicContainer**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0*) → BasicContainer

Method to set the parameter(s) of BasicContainer.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for BasicContainer.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for BasicContainer.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for BasicContainer.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for BasicContainer.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for BasicContainer.

BlinkingContainer

class **BlinkingContainer**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*BlinkingType=0, Visible=True*) → BlinkingContainer

Method to set the parameter(s) of BlinkingContainer.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for BlinkingContainer.

blinking_type(*v_blinking_type*) → Widget

Method to build runtime message A661_BLINKING_TYPE for BlinkingContainer.

BroadcastReceiver

class **BroadcastReceiver**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*NumberOfFields=1, ParameterIdent='A661_ACTIVE_TABBED_PANEL', WidgetStructure=[0]*) → BroadcastReceiver

Method to set the parameter(s) of BroadcastReceiver.

broadcast_data(*data:* Widget | *ExtensionWidget* | *Msg*) → Widget

Method to build runtime message A661_BROADCAST_DATA for BroadcastReceiver.

Arguments

data: the target runtime message, either of type Widget, ExtensionWidget, or Msg.

- When it is of type Widget or ExtensionWidget, use any widget or extension to build the message:

  <widget>(0).<target_message>(<args>)

  <extension>(0).<target_message>(<args>)

  Example: if the broadcastReceiver expects an A661_VISIBLE message, use any widet having the visible message function to send the message (widget id does not matter):

```
BroadcastReceiver(1).broadcast_data(ToggleButton(0).visible(False))
```

- When it is of type Msg, a message object corresponding to the message should be provided.

  Example for visible message:

```
BroadcastReceiver(1).broadcast_data(Msg().const('A661_VISIBLE').uch
ar(0).pad(1))
```

BufferFormat

class **BufferFormat**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*NumberOfFields=1, BufferStructure=[{'WidgetIdent': 1, 'ParameterIdent': 'A661_AC_LAT'}]*) → BufferFormat

Method to set the parameter(s) of BufferFormat.

buffer_of_param(*buffer: Msg*) → Widget

Method to build runtime message A661_BUFFER_OF_PARAM for BufferFormat.

Arguments

buffer: the buffer structure of type Msg.

The message is built with respect to buffer format structure, as described in A661 standard (BufferFormat widget section).

Example: BufferFormat with Visible and StyleSet parameters:

```
BufferFormat(1).buffer_of_param(Msg().uchar(1).pad(1).ushort(100))
```

CheckButton

class **CheckButton**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=50, NextFocusedWidget=0, MaxStringLength=32, CheckButtonState='A661_UNSELECTED', Alignment='A661_LEFT', AutomaticFocusMotion=False, PicturePosition='A661_LEFT', LabelString='LabelString'*) → CheckButton

Method to set the parameter(s) of CheckButton.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for CheckButton.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for CheckButton.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for CheckButton.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for CheckButton.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for CheckButton.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for CheckButton.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for CheckButton.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for CheckButton.

inner_state_check(*v_check_button_state: int | str | a661_selection*) → Widget

Method to build runtime message A661_INNER_STATE_CHECK for CheckButton.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for CheckButton.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for CheckButton.

checkbox_position(*v_checkbox_position: int | str | a661_checkbox_position_left_right*) → Widget

Method to build runtime message A661_CHECKBOX_POSITION for CheckButton.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for CheckButton.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for CheckButton.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for CheckButton.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for CheckButton.

ComboBox

class **ComboBox**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, SelectingAreaWidth=3000, SelectingAreaHeight=4900, StyleSet=110, NextFocusedWidget=0, MaxNumberOfEntries=16, NumberOfEntries=4, SelectedEntry=1, MaxStringLength=32, OpeningEntry=1, Alignment='A661_LEFT', OpeningMode='A661_OPEN_DOWN', AutomaticFocusMotion=False, EntryList=['Entry1', 'Entry2', 'Entry3', 'Entry4']*) → ComboBox

Method to set the parameter(s) of ComboBox.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for ComboBox.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for ComboBox.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for ComboBox.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for ComboBox.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for ComboBox.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for ComboBox.

selecting_width(*v_selecting_area_width*) → Widget

Method to build runtime message A661_SELECTING_WIDTH for ComboBox.

selecting_height(*v_selecting_area_height*) → Widget

Method to build runtime message A661_SELECTING_HEIGHT for ComboBox.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for ComboBox.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for ComboBox.

number_of_entries(*v_number_of_entries*) → Widget

Method to build runtime message A661_NUMBER_OF_ENTRIES for ComboBox.

selected_entry(*v_selected_entry*) → Widget

Method to build runtime message A661_SELECTED_ENTRY for ComboBox.

opening_entry(*v_opening_entry*) → Widget

Method to build runtime message A661_OPENING_ENTRY for ComboBox.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for ComboBox.

opening_mode(*v_opening_mode: int | str | a661_combo_opening_mode*) → Widget

Method to build runtime message A661_OPENING_MODE for ComboBox.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for ComboBox.

string_array(*v_array, v_number_of_strings=None*) → Widget

Method to build runtime message A661_STRING_ARRAY for ComboBox.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for ComboBox.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for ComboBox.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for ComboBox.

ComboBoxEdit

class **ComboBoxEdit**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, SelectingAreaWidth=3000, SelectingAreaHeight=4900, StyleSet=110, NextFocusedWidget=0, MaxNumberOfEntries=16, NumberOfEntries=4, SelectedEntry=1, OpeningEntry=1, MaxStringLength=32, OpeningMode='A661_OPEN_DOWN', AutomaticFocusMotion=False, StartCursorPos=32, ReportAllChanges='A661_EDB_CHANGE_CONFIRMED', Alignment='A661_LEFT', EntryList=['Entry1', 'Entry2', 'Entry3', 'Entry4']*) → ComboBoxEdit

Method to set the parameter(s) of ComboBoxEdit.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for ComboBoxEdit.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for ComboBoxEdit.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for ComboBoxEdit.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for ComboBoxEdit.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for ComboBoxEdit.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for ComboBoxEdit.

selecting_width(*v_selecting_area_width*) → Widget

Method to build runtime message A661_SELECTING_WIDTH for ComboBoxEdit.

selecting_height(*v_selecting_area_height*) → Widget

Method to build runtime message A661_SELECTING_HEIGHT for ComboBoxEdit.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for ComboBoxEdit.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for ComboBoxEdit.

number_of_entries(*v_number_of_entries*) → Widget

Method to build runtime message A661_NUMBER_OF_ENTRIES for ComboBoxEdit.

selected_entry(*v_selected_entry*) → Widget

Method to build runtime message A661_SELECTED_ENTRY for ComboBoxEdit.

opening_entry(*v_opening_entry*) → Widget

Method to build runtime message A661_OPENING_ENTRY for ComboBoxEdit.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for ComboBoxEdit.

opening_mode(*v_opening_mode: int | str | a661_combo_opening_mode*) → Widget

Method to build runtime message A661_OPENING_MODE for ComboBoxEdit.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for ComboBoxEdit.

cursor_pos(*v_start_cursor_pos*) → Widget

Method to build runtime message A661_CURSOR_POS for ComboBoxEdit.

string_array(*v_array, v_number_of_strings=None*) → Widget

Method to build runtime message A661_STRING_ARRAY for ComboBoxEdit.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for ComboBoxEdit.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for ComboBoxEdit.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for ComboBoxEdit.

Connector

class **Connector**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*ConnectorReference=0, Visible=True, Enable='A661_TRUE'*) → Connector

Method to set the parameter(s) of Connector.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for Connector.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for Connector.

CursorOver

class **CursorOver**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Visible=True, Enable='A661_TRUE', PosX=0, PosY=0, SizeX=2000, SizeY=600, StyleSet=0, PositionReportMode='A661_REPORT_ON_TRANSITION'*) → CursorOver

Method to set the parameter(s) of CursorOver.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for CursorOver.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for CursorOver.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for CursorOver.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for CursorOver.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for CursorOver.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for CursorOver.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for CursorOver.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for CursorOver.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for CursorOver.

CursorPosOverlay

class **CursorPosOverlay**(*widget_id: int | Counter, children: List[Widget | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', PosX=0, PosY=0, SizeX=2000, SizeY=600*) → CursorPosOverlay

Method to set the parameter(s) of CursorPosOverlay.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for CursorPosOverlay.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for CursorPosOverlay.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for CursorPosOverlay.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for CursorPosOverlay.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for CursorPosOverlay.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for CursorPosOverlay.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for CursorPosOverlay.

CursorRef

class **CursorRef**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*PosX=0, PosY=0)* → CursorRef

Method to set the parameter(s) of CursorRef.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for CursorRef.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for CursorRef.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for CursorRef.

DataConnector

class **DataConnector**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*ConnectorReference=0, Visible=True, Enable='A661_TRUE', DataLength=3, Data=[0, 0, 0]*) → DataConnector

Method to set the parameter(s) of DataConnector.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for DataConnector.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for DataConnector.

app_data(*v_data, v_size=None*) → Widget

Method to build runtime message A661_APP_DATA for DataConnector.

DataScalingFR180

class **DataScalingFR180**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*TargetWidgetID=0, TargetParameterID='A661_AC_LAT', Clamping='A661_BOTH_CLAMPED', StartValue=0.0, EndValue=1.0, StartScaledValue=0.0, EndScaledValue=45.0*) → DataScalingFR180

Method to set the parameter(s) of DataScalingFR180.

clamping(*v_clamping: int | str | a661_clamping*) → Widget

Method to build runtime message A661_CLAMPING for DataScalingFR180.

value(*v_value*) → Widget

Method to build runtime message A661_VALUE for DataScalingFR180.

start_value(*v_start_value*) → Widget

Method to build runtime message A661_START_VALUE for DataScalingFR180.

end_value(*v_end_value*) → Widget

Method to build runtime message A661_END_VALUE for DataScalingFR180.

start_scaled_value(*v_start_scaled_value*) → Widget

Method to build runtime message A661_START_SCALED_VALUE for DataScalingFR180.

end_scaled_value(*v_end_scaled_value*) → Widget

Method to build runtime message A661_END_SCALED_VALUE for DataScalingFR180.

DataScalingLong

class **DataScalingLong**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*TargetWidgetID=0, TargetParameterID='A661_BOUND_X', Clamping='A661_BOTH_CLAMPED', StartValue=0.0, EndValue=1.0, StartScaledValue=0, EndScaledValue=1000*) → DataScalingLong

Method to set the parameter(s) of DataScalingLong.

clamping(*v_clamping: int | str | a661_clamping*) → Widget

Method to build runtime message A661_CLAMPING for DataScalingLong.

value(*v_value*) → Widget

Method to build runtime message A661_VALUE for DataScalingLong.

start_value(*v_start_value*) → Widget

Method to build runtime message A661_START_VALUE for DataScalingLong.

end_value(*v_end_value*) → Widget

Method to build runtime message A661_END_VALUE for DataScalingLong.

start_scaled_value(*v_start_scaled_value*) → Widget

Method to build runtime message A661_START_SCALED_VALUE for DataScalingLong.

end_scaled_value(*v_end_scaled_value*) → Widget

Method to build runtime message A661_END_SCALED_VALUE for DataScalingLong.

DataScalingULong

class **DataScalingULong**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*TargetWidgetID=0, TargetParameterID='A661_ALPHA_MASK', Clamping='A661_BOTH_CLAMPED', StartValue=0.0, EndValue=1.0, StartScaledValue=0, EndScaledValue=1000*) → DataScalingULong

Method to set the parameter(s) of DataScalingULong.

clamping(*v_clamping: int | str | a661_clamping*) → Widget

Method to build runtime message A661_CLAMPING for DataScalingULong.

value(*v_value*) → Widget

Method to build runtime message A661_VALUE for DataScalingULong.

start_value(*v_start_value*) → Widget

Method to build runtime message A661_START_VALUE for DataScalingULong.

end_value(*v_end_value*) → Widget

Method to build runtime message A661_END_VALUE for DataScalingULong.

start_scaled_value(*v_start_scaled_value*) → Widget

Method to build runtime message A661_START_SCALED_VALUE for DataScalingULong.

end_scaled_value(*v_end_scaled_value*) → Widget

Method to build runtime message A661_END_SCALED_VALUE for DataScalingULong.

EditBoxMasked

class **EditBoxMasked**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=4000, SizeY=800, StyleSet=110, NextFocusedWidget=0, AlphaMask=3760062464, NumericMask=203292672, StartCursorPos=11, AutomaticFocusMotion=False, ReportAllChanges='A661_EDB_CHANGE_CONFIRMED', Alignment='A661_LEFT', LabelString='ABC/12 LBL:*****'*) → EditBoxMasked

Method to set the parameter(s) of EditBoxMasked.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for EditBoxMasked.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for EditBoxMasked.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for EditBoxMasked.

cursor_pos(*v_start_cursor_pos*) → Widget

Method to build runtime message A661_CURSOR_POS for EditBoxMasked.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for EditBoxMasked.

alpha_mask(*v_alpha_mask*) → Widget

Method to build runtime message A661_ALPHA_MASK for EditBoxMasked.

numeric_mask(*v_numeric_mask*) → Widget

Method to build runtime message A661_NUMERIC_MASK for EditBoxMasked.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for EditBoxMasked.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build message A661_NEXT_FOCUSED_WIDGET for EditBoxMasked.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build message A661_AUTO_FOCUS_MOTION for EditBoxMasked.

EditBoxMultiLine

class **EditBoxMultiLine**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=110, NextFocusedWidget=0, StartCursorPos=32, MaxStringLength=32, AutomaticFocusMotion=False, ReportAllChanges='A661_EDB_CHANGE_CONFIRMED', Alignment='A661_LEFT', VerticalScroll='A661_RIGHT', LabelString='LabelString'*) → EditBoxMultiLine

Method to set the parameter(s) of EditBoxMultiLine.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for EditBoxMultiLine.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for EditBoxMultiLine.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for EditBoxMultiLine.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for EditBoxMultiLine.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for EditBoxMultiLine.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for EditBoxMultiLine.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for EditBoxMultiLine.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for EditBoxMultiLine.

cursor_pos(*v_start_cursor_pos*) → Widget

Method to build runtime message A661_CURSOR_POS for EditBoxMultiLine.

auto_focus_motion(*v_automatic_focus_motion: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for EditBoxMultiLine.

alignment(*v_alignment: int* | *str* | *a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for EditBoxMultiLine.

vertical_scroll(*v_vertical_scroll: int* | *str* | *a661_align_lartb*) → Widget

Method to build runtime message A661_VERTICAL_SCROLL for EditBoxMultiLine.

string(*v_label_string*, *v_string_size=None*) → Widget

Method to build runtime message A661_STRING for EditBoxMultiLine.

entry_valid(*v_entry_validation: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for EditBoxMultiLine.

pos_xy(*v_pos_x*, *v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for EditBoxMultiLine.

size_xy(*v_size_x*, *v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for EditBoxMultiLine.

EditBoxNumeric

class **EditBoxNumeric**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=110, NextFocusedWidget=0, Value=0.0, TicsCoarse=10.0, TicsFine=0.1, MinValue=0.0, MaxValue=100.0, LegendAreaSizeX=800, StartCursorPosByte=32, MaxFormatStringLength=16, MaxLegendStringLength=16, LegendPosition='A661_RIGHT', AutomaticFocusMotion=False, ReportAllChanges='A661_EDB_CHANGE_CONFIRMED', Alignment='A661_LEFT', LegendRemoved=False, NumericKeyFlag=True, CyclicFlag=False, FormatString='+___.##', LegendString='LD'*) → EditBoxNumeric

Method to set the parameter(s) of EditBoxNumeric.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for EditBoxNumeric.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for EditBoxNumeric.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for EditBoxNumeric.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for EditBoxNumeric.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for EditBoxNumeric.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for EditBoxNumeric.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for EditBoxNumeric.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for EditBoxNumeric.

value(*v_value*) → Widget

Method to build runtime message A661_VALUE for EditBoxNumeric.

tics_coarse(*v_tics_coarse*) → Widget

Method to build runtime message A661_TICS_COARSE for EditBoxNumeric.

tics_fine(*v_tics_fine*) → Widget

Method to build runtime message A661_TICS_FINE for EditBoxNumeric.

min_value(*v_min_value*) → Widget

Method to build runtime message A661_MIN_VALUE for EditBoxNumeric.

max_value(*v_max_value*) → Widget

Method to build runtime message A661_MAX_VALUE for EditBoxNumeric.

legend_area_size_x(*v_legend_area_size_x*) → Widget

Method to build runtime message A661_LEGEND_AREA_SIZE_X for EditBoxNumeric.

cursor_pos_byte(*v_start_cursor_pos_byte*) → Widget

Method to build runtime message A661_CURSOR_POS_BYTE for EditBoxNumeric.

legend_position(*v_legend_position: int* | *str* | *a661_align_lrtb*) → Widget

Method to build runtime message A661_LEGEND_POSITION for EditBoxNumeric.

auto_focus_motion(*v_automatic_focus_motion: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for EditBoxNumeric.

alignment(*v_alignment: int* | *str* | *a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for EditBoxNumeric.

legend_removed(*v_legend_removed: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_LEGEND_REMOVED for EditBoxNumeric.

format_string(*v_format_string, v_string_size=None*) → Widget

Method to build runtime message A661_FORMAT_STRING for EditBoxNumeric.

string(*v_legend_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for EditBoxNumeric.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for EditBoxNumeric.

minmax_values(*v_min_value, v_max_value*) → Widget

Method to build runtime message A661_MINMAX_VALUES for EditBoxNumeric.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for EditBoxNumeric.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for EditBoxNumeric.

EditBoxNumericBCD

class **EditBoxNumericBCD**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define*(Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=5000, SizeY=800, LegendAreaSizeX=800, StyleSet=110, NextFocusedWidget=0, Value=0, MinValue=17400036876686524416, MaxValue=1261007895663738880, MinFieldsValues=0, MaxFieldsValues=108184632420794368, TicsCoarse=4294967296, TicsFine=5242880, NumberOfFields=4, Alignment='A661_LEFT', LegendPosition='A661_RIGHT', LegendRemoved=False, MaxFormatStringLength=32, MaxLegendStringLength=16, PositiveStringLength=16, NegativeStringLength=16, AutomaticFocusMotion=False, ReportAllChanges='A661_EDB_CHANGE_CONFIRMED', NumericKeyFlag=True, CyclicFlag=False, SizeOfFields=[3, 2, 2, 4], FormatString='###d##\'##.####"+', LegendString='LD', PositiveString='E', NegativeString='W')* → EditBoxNumericBCD

Method to set the parameter(s) of EditBoxNumericBCD.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for EditBoxNumericBCD.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for EditBoxNumericBCD.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for EditBoxNumericBCD.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for EditBoxNumericBCD.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for EditBoxNumericBCD.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for EditBoxNumericBCD.

legend_area_size_x(*v_legend_area_size_x*) → Widget

Method to build runtime message A661_LEGEND_AREA_SIZE_X for EditBoxNumericBCD.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for EditBoxNumericBCD.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for EditBoxNumericBCD.

value_8byte(*v_value*) → Widget

Method to build runtime message A661_VALUE_8BYTE for EditBoxNumericBCD.

tics_coarse_8byte(*v_tics_coarse*) → Widget

Method to build runtime message A661_TICS_COARSE_8BYTE for EditBoxNumericBCD.

tics_fine_8byte(*v_tics_fine*) → Widget

Method to build runtime message A661_TICS_FINE_8BYTE for EditBoxNumericBCD.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for EditBoxNumericBCD.

legend_position(*v_legend_position: int | str | a661_align_lrtb*) → Widget

Method to build runtime message A661_LEGEND_POSITION for EditBoxNumericBCD

legend_removed(*v_legend_removed: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_LEGEND_REMOVED for EditBoxNumericBCD.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for EditBoxNumericBCD.

format_string(*v_format_string, v_string_size=None*) → Widget

Method to build runtime message A661_FORMAT_STRING for EditBoxNumericBCD.

string(*v_legend_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for EditBoxNumericBCD.

positive_string(*v_positive_string, v_string_size=None*) → Widget

Method to build runtime message A661_POSITIVE_STRING for EditBoxNumericBCD.

negative_string(*v_negative_string, v_string_size=None*) → Widget

Method to build runtime message A661_NEGATIVE_STRING for EditBoxNumericBCD.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for EditBoxNumericBCD.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for EditBoxNumericBCD.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for EditBoxNumericBCD.

EditBoxText

class **EditBoxText**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=110, NextFocusedWidget=0, StartCursorPos=32, MaxStringLength=32, AutomaticFocusMotion=False, ReportAllChanges='A661_EDB_CHANGE_CONFIRMED', Alignment='A661_LEFT', LabelString='LabelString'*) → EditBoxText

Method to set the parameter(s) of EditBoxText.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for EditBoxText.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for EditBoxText.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for EditBoxText.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for EditBoxText.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for EditBoxText.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for EditBoxText.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for EditBoxText.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for EditBoxText.

cursor_pos(*v_start_cursor_pos*) → Widget

Method to build runtime message A661_CURSOR_POS for EditBoxText.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for EditBoxText.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for EditBoxText.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for EditBoxText.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for EditBoxText.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for EditBoxText.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for EditBoxText.

EventHandler

class **EventHandler**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*EnableHandler=True, TriggerLayerIdent=0, TriggerWidgetIdent=0, TriggerEvent='A661_EVT_STATE_CHANGE', NumWidgetParamResults=0, NumLayerReqResults=0, WidgetParamResultArray=[], LayerReqResultArray=[]*) → EventHandler

Method to set the parameter(s) of EventHandler.

enable_handler(*v_enable_handler: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENABLE_HANDLER for EventHandler.

ExternalSource

class **ExternalSource**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, SourceReference=0, SourceX=0, SourceY=0, SourceDX=500, SourceDY=300, StyleSet=1*) → ExternalSource

Method to set the parameter(s) of ExternalSource.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for ExternalSource.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for ExternalSource.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for ExternalSource.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for ExternalSource.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for ExternalSource.

source_ref(*v_source_reference*) → Widget

Method to build runtime message A661_SOURCE_REF for ExternalSource.

source_x(*v_source_x*) → Widget

Method to build runtime message A661_SOURCE_X for ExternalSource.

source_y(*v_source_y*) → Widget

Method to build runtime message A661_SOURCE_Y for ExternalSource.

source_dx(*v_source_dx*) → Widget

Method to build runtime message A661_SOURCE_DX for ExternalSource.

source_dy(*v_source_dy*) → Widget

Method to build runtime message A661_SOURCE_DY for ExternalSource.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for ExternalSource.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for ExternalSource.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for ExternalSource.

source_xy(*v_source_x, v_source_y*) → Widget

Method to build runtime message A661_SOURCE_XY for ExternalSource.

source_dxdy(*v_source_dx, v_source_dy*) → Widget

Method to build runtime message A661_SOURCE_DXDY for ExternalSource.

FocusIn

class **FocusIn**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*NextFocusedWidget=0*) → FocusIn

Method to set the parameter(s) of FocusIn.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for FocusIn.

FocusLink

class **FocusLink**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*NextLayerConnectorRef=0, NextFocusedWidget=0*) → FocusLink

Method to set the parameter(s) of FocusLink.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for FocusLink.

FocusOut

class **FocusOut**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*NextFocusInAppId=1, NextFocusInLayerId=1, NextFocusInId=0*) → FocusOut

Method to set the parameter(s) of FocusOut.

next_focus_in_app_id(*v_next_focus_in_app_id*) → Widget

Method to build runtime message A661_NEXT_FOCUS_IN_APP_ID for FocusOut.

next_focus_in_layer_id(*v_next_focus_in_layer_id*) → Widget

Method to build runtime message A661_NEXT_FOCUS_IN_LAYER_ID for FocusOut.

next_focus_in_id(*v_next_focus_in_id*) → Widget

Method to build runtime message A661_NEXT_FOCUS_IN_ID for FocusOut.

GestureArea

class **GestureArea**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, StyleSet=0, PosX=0, PosY=0, SizeX=2000, SizeY=2000, ReportMode='A661_REPORT_ON_TRANSITION', AllowOutside=False, NumberOfEvents=2, EventIdArray=['A661_GESTURE_TAP', 'A661_GESTURE_FLICK']*) → GestureArea

Method to set the parameter(s) of GestureArea.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for GestureArea.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for GestureArea.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for GestureArea.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for GestureArea.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for GestureArea.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for GestureArea.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for GestureArea.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for GestureArea.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for GestureArea.

GpArcCircle

class **GpArcCircle**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, Visible=True, PosX=0, PosY=0, StartAngle=0.0, EndAngle=90.0, Radius=2000, StyleSet=1, ColorIndex=1, Filled=True, FillIndex=74, Halo='A661_FALSE'*) → GpArcCircle

Method to set the parameter(s) of GpArcCircle.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for GpArcCircle.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for GpArcCircle.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for GpArcCircle.

start_angle(*v_start_angle*) → Widget

Method to build runtime message A661_START_ANGLE for GpArcCircle.

end_angle(*v_end_angle*) → Widget

Method to build runtime message A661_END_ANGLE for GpArcCircle.

radius(*v_radius*) → Widget

Method to build runtime message A661_RADIUS for GpArcCircle.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for GpArcCircle.

color_index(*v_color_index*) → Widget

Method to build runtime message A661_COLOR_INDEX for GpArcCircle.

filled(*v_filled: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_FILLED for GpArcCircle.

fill_index(*v_fill_index*) → Widget

Method to build runtime message A661_FILL_INDEX for GpArcCircle.

halo(*v_halo: int | str | a661_halo*) → Widget

Method to build runtime message A661_HALO for GpArcCircle.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for GpArcCircle.

GpArcEllipse

class **GpArcEllipse**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, Visible=True, PosX=0, PosY=0, SizeX=2000, SizeY=600, StartAngle=0.0, EndAngle=90.0, StyleSet=1, ColorIndex=1, Filled=True, FillIndex=74, Halo='A661_FALSE'*) → GpArcEllipse

Method to set the parameter(s) of GpArcEllipse.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for GpArcEllipse.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for GpArcEllipse.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for GpArcEllipse.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for GpArcEllipse.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for GpArcEllipse.

start_angle(*v_start_angle*) → Widget

Method to build runtime message A661_START_ANGLE for GpArcEllipse.

end_angle(*v_end_angle*) → Widget

Method to build runtime message A661_END_ANGLE for GpArcEllipse.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for GpArcEllipse.

color_index(*v_color_index*) → Widget

Method to build runtime message A661_COLOR_INDEX for GpArcEllipse.

filled(*v_filled: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_FILLED for GpArcEllipse.

fill_index(*v_fill_index*) → Widget

Method to build runtime message A661_FILL_INDEX for GpArcEllipse.

halo(*v_halo: int* | *str* | *a661_halo*) → Widget

Method to build runtime message A661_HALO for GpArcEllipse.

pos_xy(*v_pos_x*, *v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for GpArcEllipse.

size_xy(*v_size_x*, *v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for GpArcEllipse.

GpCrown

class **GpCrown**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, Visible=True, PosX=0, PosY=0, StartAngle=0.0, EndAngle=90.0, InnerRadius=1500, OuterRadius=2000, StyleSet=1, ColorIndex=1, Filled=True, FillIndex=74, Halo='A661_FALSE'*) → GpCrown

Method to set the parameter(s) of GpCrown.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for GpCrown.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for GpCrown.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for GpCrown.

start_angle(*v_start_angle*) → Widget

Method to build runtime message A661_START_ANGLE for GpCrown.

end_angle(*v_end_angle*) → Widget

Method to build runtime message A661_END_ANGLE for GpCrown.

inner_radius(*v_inner_radius*) → Widget

Method to build runtime message A661_INNER_RADIUS for GpCrown.

outer_radius(*v_outer_radius*) → Widget

Method to build runtime message A661_OUTER_RADIUS for GpCrown.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for GpCrown.

color_index(*v_color_index*) → Widget

Method to build runtime message A661_COLOR_INDEX for GpCrown.

filled(*v_filled: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_FILLED for GpCrown.

fill_index(*v_fill_index*) → Widget

Method to build runtime message A661_FILL_INDEX for GpCrown.

halo(*v_halo: int* | *str* | *a661_halo*) → Widget

Method to build runtime message A661_HALO for GpCrown.

pos_xy(*v_pos_x*, *v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for GpCrown.

GpLine

class **GpLine**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, Visible=True, PosXStart=0, PosYStart=0, PosXEnd=1000, PosYEnd=1000, StyleSet=1, ColorIndex=1, Halo='A661_FALSE'*) → GpLine

Method to set the parameter(s) of GpLine.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for GpLine.

pos_x(*v_pos_xstart*) → Widget

Method to build runtime message A661_POS_X for GpLine.

pos_y(*v_pos_ystart*) → Widget

Method to build runtime message A661_POS_Y for GpLine.

pos_x2(*v_pos_xend*) → Widget

Method to build runtime message A661_POS_X2 for GpLine.

pos_y2(*v_pos_yend*) → Widget

Method to build runtime message A661_POS_Y2 for GpLine.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for GpLine.

color_index(*v_color_index*) → Widget

Method to build runtime message A661_COLOR_INDEX for GpLine.

halo(*v_halo: int | str | a661_halo*) → Widget

Method to build runtime message A661_HALO for GpLine.

pos_xy(*v_pos_xstart, v_pos_ystart*) → Widget

Method to build runtime message A661_POS_XY for GpLine.

pos_xy2(*v_pos_xend, v_pos_yend*) → Widget

Method to build runtime message A661_POS_XY2 for GpLine.

GpLinePolar

class **GpLinePolar**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, Visible=True, PosXStart=0, PosYStart=0, RotationAngle=45.0, LineLength=1000, StyleSet=1, ColorIndex=1, Halo='A661_FALSE'*) → GpLinePolar

Method to set the parameter(s) of GpLinePolar.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for GpLinePolar.

pos_x(*v_pos_xstart*) → Widget

Method to build runtime message A661_POS_X for GpLinePolar.

pos_y(*v_pos_ystart*) → Widget

Method to build runtime message A661_POS_Y for GpLinePolar.

rotation_angle(*v_rotation_angle*) → Widget

Method to build runtime message A661_ROTATION_ANGLE for GpLinePolar.

line_length(*v_line_length*) → Widget

Method to build runtime message A661_LINE_LENGTH for GpLinePolar.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for GpLinePolar.

color_index(*v_color_index*) → Widget

Method to build runtime message A661_COLOR_INDEX for GpLinePolar.

halo(*v_halo: int | str | a661_halo*) → Widget

Method to build runtime message A661_HALO for GpLinePolar.

pos_xy(*v_pos_xstart, v_pos_ystart*) → Widget

Method to build runtime message A661_POS_XY for GpLinePolar.

GpPolyline

class **GpPolyline**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, Visible=True, StyleSet=1, ColorIndex=1, Halo='A661_FALSE', Closed=False, MaxNumberOfVertices=16, NumberOfVertices=4, Vertices=[{'x': 0, 'y': 0}, {'x': 2800, 'y': 2200}, {'x': 5000, 'y': 1200}, {'x': 3600, 'y': 500}]*) → GpPolyline

Method to set the parameter(s) of GpPolyline.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for GpPolyline.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for GpPolyline.

color_index(*v_color_index*) → Widget

Method to build runtime message A661_COLOR_INDEX for GpPolyline.

halo(*v_halo: int | str | a661_halo*) → Widget

Method to build runtime message A661_HALO for GpPolyline.

closed(*v_closed: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_CLOSED for widget GpPolyline.

number_of_entries(*v_number_of_vertices*) → Widget

Method to build runtime message A661_NUMBER_OF_ENTRIES for GpPolyline.

vertices_array(*v_array, v_number_of_entries_updated=None*) → Widget

Method to build runtime message A661_VERTICES_ARRAY for GpPolyline.

GpRectangle

class **GpRectangle**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, Visible=True, PosX=0, PosY=0, SizeX=2000, SizeY=600, StyleSet=1, ColorIndex=1, Filled=True, FillIndex=74, Halo='A661_FALSE'*) → GpRectangle

Method to set the parameter(s) of GpRectangle.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for GpRectangle.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for GpRectangle.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for GpRectangle.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for GpRectangle.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for GpRectangle.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for GpRectangle.

color_index(*v_color_index*) → Widget

Method to build runtime message A661_COLOR_INDEX for GpRectangle.

filled(*v_filled: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_FILLED for GpRectangle.

fill_index(*v_fill_index*) → Widget

Method to build runtime message A661_FILL_INDEX for GpRectangle.

halo(*v_halo: int | str | a661_halo*) → Widget

Method to build runtime message A661_HALO for GpRectangle.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for GpRectangle.

size_xy(*v_size_x*, *v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for GpRectangle.

GpTriangle

class **GpTriangle**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, Visible=True, PosX=0, PosY=0, PosX2=600, PosY2=1000, PosX3=1200, PosY3=0, StyleSet=1, ColorIndex=1, Filled=True, FillIndex=74, Halo='A661_FALSE'*) → GpTriangle

Method to set the parameter(s) of GpTriangle.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for GpTriangle.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for GpTriangle.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for GpTriangle.

pos_x2(*v_pos_x2*) → Widget

Method to build runtime message A661_POS_X2 for GpTriangle.

pos_y2(*v_pos_y2*) → Widget

Method to build runtime message A661_POS_Y2 for GpTriangle.

pos_x3(*v_pos_x3*) → Widget

Method to build runtime message A661_POS_X3 for GpTriangle.

pos_y3(*v_pos_y3*) → Widget

Method to build runtime message A661_POS_Y3 for GpTriangle.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for GpTriangle.

color_index(*v_color_index*) → Widget

Method to build runtime message A661_COLOR_INDEX for GpTriangle.

filled(*v_filled: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_FILLED for GpTriangle.

fill_index(*v_fill_index*) → Widget

Method to build runtime message A661_FILL_INDEX for GpTriangle.

halo(*v_halo: int | str | a661_halo*) → Widget

Method to build runtime message A661_HALO for GpTriangle.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for GpTriangle.

pos_xy2(*v_pos_x2, v_pos_y2*) → Widget

Method to build runtime message A661_POS_XY2 for GpTriangle.

pos_xy3(*v_pos_x3, v_pos_y3*) → Widget

Method to build runtime message A661_POS_XY3 for GpTriangle.

InkArea

class **InkArea**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Visible=True, Enable='A661_TRUE', StyleSet=1, PosX=0, PosY=0, SizeX=2000, SizeY=2000, ColorIndex=21, Filled=True, FillIndex=74, MaxBufferSize=1000, NextFocusedWidget=0*) → InkArea

Method to set the parameter(s) of InkArea.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for InkArea.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for InkArea.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for InkArea.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for InkArea.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for InkArea.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for InkArea.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for InkArea.

color_index(*v_color_index*) → Widget

Method to build runtime message A661_COLOR_INDEX for InkArea.

filled(*v_filled: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_FILLED for InkArea.

fill_index(*v_fill_index*) → Widget

Method to build runtime message A661_FILL_INDEX for InkArea.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for InkArea.

clear(*v_clear: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_CLEAR for InkArea.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for InkArea.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for InkArea.

KeyboardArea

class **KeyboardArea**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', NumberOfKeys=2, MaxNumberOfKeys=10, PosX=0, PosY=0, SizeX=2000, SizeY=2000, KeyArray=[65, 321]*) → KeyboardArea

Method to set the parameter(s) of KeyboardArea.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for KeyboardArea.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for KeyboardArea.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for KeyboardArea.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for KeyboardArea.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for KeyboardArea.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for KeyboardArea.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for KeyboardArea.

number_of_keys(*v_number_of_keys*) → Widget

Method to build runtime message A661_NUMBER_OF_KEYS for KeyboardArea.

key_array(*v_array, v_number_of_entries=None*) → Widget

Method to build runtime message A661_KEY_ARRAY for KeyboardArea.

Label

class **Label**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, RotationAngle=0.0, StyleSet=1, MaxStringLength=32, MotionAllowed=True, Font=0, ColorIndex=1, Alignment='A661_LEFT', LabelString='LabelString'*) → Label

Method to set the parameter(s) of Label.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for Label.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for Label.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for Label.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for Label.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for Label.

orientation(*v_rotation_angle*) → Widget

Method to build runtime message A661_ORIENTATION for Label.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for Label.

font(*v_font*) → Widget

Method to build runtime message A661_FONT for Label.

color_index(*v_color_index*) → Widget

Method to build runtime message A661_COLOR_INDEX for Label.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for Label.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for Label.

pos_xy(*v_pos_x*, *v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for Label.

size_xy(*v_size_x*, *v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for Label.

LabelComplex

class **LabelComplex**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=50, MaxStringLength=32, Alignment='A661_LEFT', DefaultStyleText=[27, 64, 51, 27, 65, 21, 27, 66, 41, 27, 67, 1], LabelString='Label\x1bDComp\x1bE\x1bAJlex'*) → LabelComplex

Method to set the parameter(s) of LabelComplex.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for LabelComplex.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for LabelComplex.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for LabelComplex.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for LabelComplex.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for LabelComplex.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for LabelComplex.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for LabelComplex.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for LabelComplex.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for Label.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for Label.

MapBoundary

class **MapBoundary**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, MaxNumberOfEntries=255, NumberOfEntries=4, BeginAngleArray=[-45.0, 0.0, 179.99999991618097, -135.0], DistanceArray=[1350, 1350, 1350, 1350], AlgorithmArray=['A661_VERT_LINE', 'A661_ARC', 'A661_VERT_LINE', 'A661_HORZ_LINE']*) → MapBoundary

Method to set the parameter(s) of MapBoundary.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for MapBoundary.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MapBoundary.

number_of_entries(*v_number_of_entries*) → Widget

Method to build runtime message A661_NUMBER_OF_ENTRIES for MapBoundary.

boundary_array(*v_array, v_number_of_entries_updated=None*) → Widget

Method to build runtime message A661_BOUNDARY_ARRAY for MapBoundary.

MapGrid

class **MapGrid**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Visible=True, OffsetX=0.0, OffsetY=0.0, IncrementX=0.0, IncrementY=0.0, CountX=256, CountY=256*) → MapGrid

Method to set the parameter(s) of MapGrid.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MapGrid.

mapgrid_offset(*v_offset_x, v_offset_y*) → Widget

Method to build runtime message A661_MAPGRID_OFFSET for MapGrid.

mapgrid_cellsize(*v_increment_x, v_increment_y*) → Widget

Method to build runtime message A661_MAPGRID_CELLSIZE for MapGrid.

buffer_of_fill_styles(*start_index_x: int, start_index_y: int, num_columns: int, num_rows: int, step_x: int, step_y: int, row_major: int | str | bool | a661_bool, control_flag: int, parameter_value: List[int]| Msg*) → Widget

Method to build runtime message A661_BUFFER_OF_FILL_STYLES for MapGrid.

Arguments

start_index_x: Index (zero-based) of column to start storing data.

start_index_y: Index (zero-based) of row to start storing data.

num_columns: Number of columns being filled.

num_rows: Number of rows being filled.

step_x: +1 or -1 (0xFF). May be set to 0 if *num_columns* is 1.

step_y: +1 or -1 (0xFF). May be set to 0 if *num_rows* is 1.

row_major: If True (A661_TRUE), the second Fill Style Index in this message goes into the same COLUMN as the first. If False (A661_FALSE), the second one goes into the same ROW as the first.

control_flag: bit 0 = clear buffer, bit 1 = buffer complete

parameter_value: List of fill style index values. It can either be a list of integers or an Msg object.

map_synchronization_number(*v_map_synchronization_number*) → Widget

Method to build runtime message A661_MAP_SYNCHRONIZATION_NUMBER for MapGrid.

MapHorz

class **MapHorz**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=2000, SizeY=2000, ScreenReferencePointX=0, ScreenReferencePointY=0, Range=100.0, ScreenRange=2000, ProjectionType=0*) → MapHorz

Method to set the parameter(s) of MapHorz.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for MapHorz.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MapHorz.

prp_lat(*v_projection_reference_point_latitude*) → Widget

Method to build runtime message A661_PRP_LAT for MapHorz.

prp_lat_long(*v_projection_reference_point_latitude, v_projection_reference_point_longitude*) → Widget

Method to build runtime message A661_PRP_LAT_LONG for MapHorz.

prp_long(*v_projection_reference_point_longitude*) → Widget

Method to build runtime message A661_PRP_LONG for MapHorz.

prp_screen_x(*v_screen_reference_point_x*) → Widget

Method to build runtime message A661_PRP_SCREEN_X for MapHorz.

prp_screen_xy(*v_screen_reference_point_x, v_screen_reference_point_y*) → Widget

Method to build runtime message A661_PRP_SCREEN_XY for MapHorz.

prp_screen_y(*v_screen_reference_point_y*) → Widget

Method to build runtime message A661_PRP_SCREEN_Y for MapHorz.

range(*v_range*) → Widget

Method to build runtime message A661_RANGE for MapHorz.

screen_range(*v_screen_range*) → Widget

Method to build runtime message A661_SCREEN_RANGE for MapHorz.

orientation(*v_orientation*) → Widget

Method to build runtime message A661_ORIENTATION for MapHorz.

ac_lat(*v_aircraft_latitude*) → Widget

Method to build runtime message A661_AC_LAT for MapHorz.

ac_long(*v_aircraft_longitude*) → Widget

Method to build runtime message A661_AC_LONG for MapHorz.

ac_lat_long(*v_aircraft_latitude, v_aircraft_longitude*) → Widget

Method to build runtime message A661_AC_LAT_LONG for MapHorz.

ac_orientation(*v_aircraft_orientation*) → Widget

Method to build runtime message A661_AC_ORIENTATION for MapHorz.

map_synchronization_number(*v_map_synchronization_number*) → Widget

Method to build runtime message A661_MAP_SYNCHRONIZATION_NUMBER for MapHorz.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for MapHorz.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for MapHorz.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for MapHorz.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for MapHorz.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for MapHorz.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for MapHorz.

MapHorz_Container

class **MapHorz_Container**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Visible=True*) → MapHorz_Container

Method to set the parameter(s) of MapHorz_Container.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MapHorz_Container.

ref_pos_xy(*v_x, v_y*) → Widget

Method to build runtime message A661_REF_POS_XY for MapHorz_Container.

ref_pos_x(*v_x*) → Widget

Method to build runtime message A661_REF_POS_X for MapHorz_Container.

ref_pos_y(*v_y*) → Widget

Method to build runtime message A661_REF_POS_Y for MapHorz_Container.

orientation(*v_orientation*) → Widget

Method to build runtime message A661_ORIENTATION for MapHorz_Container.

range(*v_range*) → Widget

Method to build runtime message A661_RANGE for MapHorz_Container.

screen_range(*v_screen_range*) → Widget

Method to build runtime message A661_SCREEN_RANGE for MapHorz_Container.

map_synchronization_number(*v_map_synchronization_number*) → Widget

Method to build runtime message A661_MAP_SYNCHRONIZATION_NUMBER for MapHorz_Container.

MapHorz_ItemList

class **MapHorz_ItemList**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, MaxNumberOfItem=255*) → MapHorz_ItemList

Method to set the parameter(s) of MapHorz_ItemList.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for MapHorz_ItemList.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MapHorz_ItemList.

buffer_of_mapitem(*clear_flag: bool, buffer: ItemListBuffer | Msg*) → Widget

Method to build runtime message A661_BUFFER_OF_MAPITEM for MapHorz_ItemList.

Arguments

clear_flag: If True, all Items will be set to NOT_USED before setting the specified Items.

buffer: Items structures (ItemListBuffer object or raw data as an Msg object).

The ItemListBuffer contains the list of items to send to the MapHorz_ItemList widget.

The class defines one function per item, whose name is add_<item> (item name is in lower case), with, for each item, the argument corresponding to the item structure.

Example: Set item A661_SYMBOL_GENERIC at coordinates 1000, 1000:

```
MapHorz_ItemList(1).buffer_of_mapitem(ItemListBuffer().add_symbol_generic(True,1,False,1000,1000,False))
```

(item index is automatically calculated if not provided)

blocked_buffer_of_mapitem(*clear_flag: bool, last_block: bool, update_number: bool, block_number: bool, buffer: ItemListBuffer | Msg*) → Widget

Method to build runtime message A661_BLOCKED_BUFFER_OF_MAPITEM for MapHorz_ItemList.

Arguments

clear_flag: If True, all Items will be set to NOT_USED before setting the specified Items.

last_block: When set to True (A661_TRUE), indicates the end of the blocked buffer command.

update_number: Unique number to indicate the sequence to which this block belongs.

block_number: Number indicating the block's position in the sequence.

buffer: Items structures (ItemListBuffer object or raw data as an Msg object).

map_synchronization_number(*v_map_synchronization_number*) → Widget

Method to build runtime message A661_MAP_SYNCHRONIZATION_NUMBER for MapHorz_ItemList.

entry_valid(*v_entry_validation: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for MapHorz_ItemList.

MapHorz_Panel

class **MapHorz_Panel**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, X=0, Y=0, SizeX=3000, SizeY=3000, StyleSet=100*) → MapHorz_Panel

Method to set the parameter(s) of MapHorz_Panel.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for MapHorz_Panel.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MapHorz_Panel.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for MapHorz_Panel.

ref_pos_xy(*v_x, v_y*) → Widget

Method to build runtime message A661_REF_POS_XY for MapHorz_Panel.

ref_pos_x(*v_x*) → Widget

Method to build runtime message A661_REF_POS_X for MapHorz_Panel.

ref_pos_y(*v_y*) → Widget

Method to build runtime message A661_REF_POS_Y for MapHorz_Panel.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for MapHorz_Panel.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for MapHorz_Panel.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for MapHorz_Panel.

map_synchronization_number(*v_map_synchronization_number*) → Widget

Method to build runtime message A661_MAP_SYNCHRONIZATION_NUMBER for MapHorz_Panel.

MapHorz_Source

class **MapHorz_Source**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, MapDataFormat='A661_MDF_LAT_LONG', EventFlag='A661_EVENT_NONE'*) → MapHorz_Source

Method to set the parameter(s) of MapHorz_Source.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for MapHorz_Source.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MapHorz_Source.

event_flag(*v_event_flag: int | str | a661_EventFlag_type*) → Widget

Method to build runtime message A661_EVENT_FLAG for MapHorz_Source.

crp_lat(*v_coordinate_reference_point_latitude*) → Widget

Method to build runtime message A661_CRP_LAT for MapHorz_Source.

crp_long(*v_coordinate_reference_point_longitude*) → Widget

Method to build runtime message A661_CRP_LONG for MapHorz_Source.

crp_lat_long(*v_coordinate_reference_point_latitude, v_coordinate_reference_point_longitude*) → Widget

Method to build runtime message A661_CRP_LAT_LONG for MapHorz_Source.

MapHorz_VertexBuffer

class **MapHorz_VertexBuffer**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, MaxNumberOfVertices=255, NumberOfVertices=12, Vertices=[{'x': 1000, 'y': 5000}, {'x': 3000, 'y': 5000}, {'x': 5000, 'y': 5000}, {'x': 4423, 'y': 4000}, {'x': 3845, 'y': 3000}, {'x': 4423, 'y': 2000}, {'x': 5000, 'y': 1000}, {'x': 3000, 'y': 1000}, {'x': 1000, 'y': 1000}, {'x': 1577, 'y': 2000}, {'x': 2145, 'y': 3000}, {'x': 1577, 'y': 4000}]*) → MapHorz_VertexBuffer

Method to set the parameter(s) of MapHorz_VertexBuffer.

number_of_entries(*v_number_of_vertices*) → Widget

Method to build runtime message A661_NUMBER_OF_ENTRIES for MapHorz_VertexBuffer.

vertices_array(*v_array, v_number_of_entries_updated=None*) → Widget

Method to build runtime message A661_VERTICES_ARRAY for MapHorz_VertexBuffer.

map_synchronization_number(*v_map_synchronization_number*) → Widget

Method to build runtime message A661_MAP_SYNCHRONIZATION_NUMBER for MapHorz_VertexBuffer.

MapVert

class **MapVert**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=2000, SizeY=2000, RangeX=100.0, RangeY=10000, RefPosX=100, RefPosY=100, RefGeoPosX=0.0, RefGeoPosY=0*) → MapVert

Method to set the parameter(s) of MapVert.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MapVert.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for MapVert.

range_x(*v_range_x*) → Widget

Method to build runtime message A661_RANGE_X for MapVert.

range_y(*v_range_y*) → Widget

Method to build runtime message A661_RANGE_Y for MapVert.

range_xy(*v_range_x, v_range_y*) → Widget

Method to build runtime message A661_RANGE_XY for MapVert.

prp_screen_x(*v_ref_pos_x*) → Widget

Method to build runtime message A661_PRP_SCREEN_X for MapVert.

prp_screen_y(*v_ref_pos_y*) → Widget

Method to build runtime message A661_PRP_SCREEN_Y for MapVert.

prp_screen_xy(*v_ref_pos_x, v_ref_pos_y*) → Widget

Method to build runtime message A661_PRP_SCREEN_XY for MapVert.

prp_x(*v_ref_geo_pos_x*) → Widget

Method to build runtime message A661_PRP_X for MapVert.

prp_y(*v_ref_geo_pos_y*) → Widget

Method to build runtime message A661_PRP_Y for MapVert.

prp_xy(*v_ref_geo_pos_x, v_ref_geo_pos_y*) → Widget

Method to build runtime message A661_PRP_XY for MapVert.

map_synchronization_number(*v_map_synchronization_number*) → Widget

Method to build runtime message A661_MAP_SYNCHRONIZATION_NUMBER for MapVert.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for MapVert.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for MapVert.

pos_xy(*v_pos_x*, *v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for MapVert.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for MapVert.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for MapVert.

size_xy(*v_size_x*, *v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for MapVert.

MapVert_Container

class **MapVert_Container**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Visible=True*) → MapVert_Container

Method to set the parameter(s) of MapVert_Container.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MapVert_Container.

ref_pos_x(*v_x*) → Widget

Method to build runtime message A661_REF_POS_X for MapVert_Container.

ref_pos_y(*v_y*) → Widget

Method to build runtime message A661_REF_POS_Y for MapVert_Container.

ref_pos_xy(*v_x, v_y*) → Widget

Method to build runtime message A661_REF_POS_XY for MapVert_Container.

range_x(*v_range_x*) → Widget

Method to build runtime message A661_RANGE_X for MapVert_Container.

range_y(*v_range_y*) → Widget

Method to build runtime message A661_RANGE_Y for MapVert_Container.

range_xy(*v_range_x, v_range_y*) → Widget

Method to build runtime message A661_RANGE_XY for MapVert_Container.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for MapVert_Container.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for MapVert_Container.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for MapVert_Container.

map_synchronization_number(*v_map_synchronization_number*) → Widget

Method to build runtime message A661_MAP_SYNCHRONIZATION_NUMBER for MapVert_Container.

MapVert_ItemList

class **MapVert_ItemList**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, MaxNumberOfItem=255*) → MapVert_ItemList

Method to set the parameter(s) of MapVert_ItemList.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for MapVert_ItemList.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MapVert_ItemList.

buffer_of_mapvert_items(*clear_flag: bool, buffer: ItemListBuffer | Msg*) → Widget

Method to build runtime message A661_BUFFER_OF_MAPVERT_ITEMS for MapVert_ItemList.

Arguments

clear_flag: If True, all Items will be set to NOT_USED before setting the specified Items.

buffer: Items structures (ItemListBuffer object or raw data as an Msg object).

The ItemListBuffer contains the list of items to send to the MapVert_ItemList widget.

The class defines one function per item, which name is add_<item> (item name is in lower case), with for each item the argument corresponding to the item structure.

Example: Set item A661_SYMBOL_GENERIC at coordinates 1000, 1000:

```
MapVert_ItemList(1).buffer_of_mapitem(ItemListBuffer().add_
symbol_generic(True,1,False,1000,1000,False))
```

(item index is automatically calculated if not provided)

blocked_buffer_of_mapvert_items(*clear_flag: bool, last_block: bool, update_number: bool, block_number: bool, buffer: ItemListBuffer | Msg*) → Widget

Method to build runtime message A661_BLOCKED_BUFFER_OF_MAPVERT_ITEMS for MapHorz_ItemList.

Arguments

clear_flag: If True, all Items will be set to NOT_USED before setting the specified Items.

last_block: When set to True (A661_TRUE), indicates the end of the blocked buffer command.

update_number: Unique number to indicate the sequence to which this block belongs.

block_number: Number indicating the block's position in the sequence.

buffer: Items structures (ItemListBuffer object or raw data as an Msg object).

map_synchronization_number(*v_map_synchronization_number*) → Widget

Method to build runtime message A661_MAP_SYNCHRONIZATION_NUMBER for MapVert_ItemList.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for MapVert_ItemList.

MapVert_Panel

class **MapVert_Panel**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, X=0, Y=0, SizeX=3000, SizeY=3000, StyleSet=100*) → MapVert_Panel

Method to set the parameter(s) of MapVert_Panel.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for MapVert_Panel.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MapVert_Panel.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for MapVert_Panel.

ref_pos_xy(*v_x*, *v_y*) → Widget

Method to build runtime message A661_REF_POS_XY for MapVert_Panel.

ref_pos_x(*v_x*) → Widget

Method to build runtime message A661_REF_POS_X for MapVert_Panel.

ref_pos_y(*v_y*) → Widget

Method to build runtime message A661_REF_POS_Y for MapVert_Panel.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for MapVert_Panel.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for MapVert_Panel.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for MapVert_Panel.

map_synchronization_number(*v_map_synchronization_number*) → Widget

Method to build runtime message A661_MAP_SYNCHRONIZATION_NUMBER for MapVert_Panel.

MapVert_Source

class **MapVert_Source**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, MapDataFormatX='A661_MDF_ABSOLUTE', MapDataFormatY='A661_MDF_ABSOLUTE', EventFlag=True*) → MapVert_Source

Method to set the parameter(s) of MapVert_Source.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for MapVert_Source.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MapVert_Source.

event_flag(*v_event_flag: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_EVENT_FLAG for MapVert_Source.

MaskContainer

class **MaskContainer**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*MaskEnabled=True, Visible=True, PosX=0, PosY=0, MaskReference=1*) → MaskContainer

Method to set the parameter(s) of MaskContainer.

mask_enabled(*v_mask_enabled: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_MASK_ENABLED for MaskContainer.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MaskContainer.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for MaskContainer.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for MaskContainer.

mask_reference(*v_mask_reference*) → Widget

Method to build runtime message A661_MASK_REFERENCE for MaskContainer.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for MaskContainer.

MenuBar

class **MenuBar**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, Horizontal=True, PosX=0, PosY=0, ButtonPos=0, ButtonSize=800*) → MenuBar

Method to set the parameter(s) of MenuBar.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for MenuBar.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MenuBar.

menu_horizontal(*v_horizontal: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_MENU_HORIZONTAL for MenuBar.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for MenuBar.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for MenuBar.

button_pos(*v_button_pos*) → Widget

Method to build runtime message A661_BUTTON_POS for MenuBar.

button_size(*v_button_size*) → Widget

Method to build runtime message A661_BUTTON_SIZE for MenuBar.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for MenuBar.

MultiStateButton

class **MultiStateButton**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=110, NextFocusedWidget=0, MaxStringLength=32, ButtonState=1, AutomaticFocusMotion=False, Alignment='A661_LEFT', MaxNumberOfStates=10, NumberOfStates=3, LabelStringArray=['State1', 'State2', 'State3']*) → MultiStateButton

Method to set the parameter(s) of MultiStateButton.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for MultiStateButton.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MultiStateButton.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for MultiStateButton.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for MultiStateButton.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for MultiStateButton.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for MultiStateButton.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for MultiStateButton.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for MultiStateButton.

multi_state_value(*v_button_state*) → Widget

Method to build runtime message A661_MULTI_STATE_VALUE for MultiStateButton.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for MultiStateButton.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for MultiStateButton.

number_of_states(*v_number_of_states*) → Widget

Method to build runtime message A661_NUMBER_OF_STATES for MultiStateButton.

string_array(*v_array, v_number_of_strings=None*) → Widget

Method to build runtime message A661_STRING_ARRAY for MultiStateButton.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for MultiStateButton.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for MultiStateButton.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for MultiStateButton.

MutuallyExclusiveContainer

class **MutuallyExclusiveContainer**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, VisibleChild=0*) → MutuallyExclusiveContainer

Method to set the parameter(s) of MutuallyExclusiveContainer.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for MutuallyExclusiveContainer.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for MutuallyExclusiveContainer.

visible_child(*v_visible_child*) → Widget

Method to build runtime message A661_VISIBLE_CHILD for MutuallyExclusiveContainer.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for MutuallyExclusiveContainer.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for MutuallyExclusiveContainer.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for MutuallyExclusiveContainer.

NoServiceMonitor

class **NoServiceMonitor**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*ShowNoServiceIdent=0*) → NoServiceMonitor

Method to set the parameter(s) of NoServiceMonitor.

NumericReadout

class **NumericReadout**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Visible=True, MotionAllowed=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, RotationAngle=0.0, StyleSet=50, Alignment='A661_LEFT', Value=0.0, LegendAreaSizeX=800, MaxFormatStringLength=16, MaxLegendStringLength=16, LegendPosition='A661_RIGHT', FormatString='+___.##', LegendString='LD'*) → NumericReadout

Method to set the parameter(s) of NumericReadout.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for NumericReadout.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for NumericReadout.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for NumericReadout.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for NumericReadout.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for NumericReadout.

orientation(*v_rotation_angle*) → Widget

Method to build runtime message A661_ORIENTATION for NumericReadout.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for NumericReadout.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for NumericReadout.

value(*v_value*) → Widget

Method to build runtime message A661_VALUE for NumericReadout.

legend_area_size_x(*v_legend_area_size_x*) → Widget

Method to build runtime message A661_LEGEND_AREA_SIZE_X for NumericReadout.

legend_position(*v_legend_position: int | str | a661_align_lrtb*) → Widget

Method to build runtime message A661_LEGEND_POSITION for NumericReadout.

format_string(*v_format_string, v_string_size=None*) → Widget

Method to build runtime message A661_FORMAT_STRING for NumericReadout.

string(*v_legend_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for NumericReadout.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for NumericReadout.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for NumericReadout.

PagingContainer

class **PagingContainer**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=3000, WrappingType='A661_WRAP_BOTH', PagingControlPosition='A661_TOP_RIGHT', FinePageDelta=1, CoarsePageDelta=5, VisibleChild=1, ReportVisibleChild=True, StyleSet=100, Anonymous=False*) → PagingContainer

Method to set the parameter(s) of PagingContainer.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for PagingContainer.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for PagingContainer.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for PagingContainer.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for PagingContainer.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for PagingContainer.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for PagingContainer.

page_wrap_type(*v_wrapping_type: int | str | a661_wrapping_type*) → Widget

Method to build runtime message A661_PAGE_WRAP_TYPE for PagingContainer.

page_control_position(*v_paging_control_position: int | str | a661_paging_control_position*) → Widget

Method to build runtime message A661_PAGE_CONTROL_POSITION for PagingContainer.

fine_page_delta(*v_fine_page_delta*) → Widget

Method to build runtime message A661_FINE_PAGE_DELTA for PagingContainer.

coarse_page_delta(*v_coarse_page_delta*) → Widget

Method to build runtime message A661_COARSE_PAGE_DELTA for PagingContainer.

visible_child(*v_visible_child*) → Widget

Method to build runtime message A661_VISIBLE_CHILD for PagingContainer.

visible_child(*v_visible_child*) → Widget

Method to build runtime message A661_VISIBLE_CHILD for PagingContainer.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for PagingContainer.

pos_xy(*v_pos_x*, *v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for PagingContainer.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for PagingContainer.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for PagingContainer.

Panel

class **Panel**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=3000, StyleSet=100, MotionAllowed=True*) → Panel

Method to set the parameter(s) of Panel.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for Panel.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for Panel.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for Panel.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for Panel.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for Panel.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for Panel.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for Panel.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for Panel.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for Panel.

Picture

class **Picture**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Anonymous=False, Visible=True, PosX=0, PosY=0, SizeX=1000, SizeY=1000, StyleSet=0, PictureReference=1*) → Picture

Method to set the parameter(s) of Picture.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for Picture.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for Picture.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for Picture.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for Picture.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for Picture.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for Picture.

picture_reference(*v_picture_reference*) → Widget

Method to build runtime message A661_PICTURE_REFERENCE for Picture.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for Picture.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for Picture.

PictureAnimated

class **PictureAnimated**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Visible=True, PosX=0, PosY=0, SizeX=1000, SizeY=1000, StyleSet=0, NumberOfPictures=4, IndexOfFrequency=10, LoopFlag=True, AnimationFlag=True, PictureArray=[11, 12, 13, 14]*) → PictureAnimated

Method to set the parameter(s) of PictureAnimated.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for PictureAnimated.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for PictureAnimated.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for PictureAnimated.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for PictureAnimated.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for PictureAnimated.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for PictureAnimated.

index_of_frequency(*v_index_of_frequency*) → Widget

Method to build message A661_INDEX_OF_FREQUENCY for PictureAnimated.

loop_flag(*v_loop_flag: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_LOOP_FLAG for PictureAnimated.

animation_flag(*v_animation_flag: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ANIMATION_FLAG for PictureAnimated.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for Picture.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for Picture.

PicturePushButton

class **PicturePushButton**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=110, NextFocusedWidget=0, PictureReference=1, MaxStringLength=32, PicturePosition='A661_RIGHT', AutomaticFocusMotion=False, Alignment='A661_LEFT', LabelString='LabelString'*) → PicturePushButton

Method to set the parameter(s) of PicturePushButton.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for PicturePushButton.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for PicturePushButton.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for PicturePushButton.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for PicturePushButton.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for PicturePushButton.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for PicturePushButton.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for PicturePushButton.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for PicturePushButton.

picture_reference(*v_picture_reference*) → Widget

Method to build runtime message A661_PICTURE_REFERENCE for PicturePushButton.

picture_position(*v_picture_position: int | str | a661_align_lcrtb*) → Widget

Method to build runtime message A661_PICTURE_POSITION for PicturePushButton.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for PicturePushButton.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for PicturePushButton.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for PicturePushButton.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for PicturePushButton.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for PicturePushButton.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for PicturePushButton.

PictureToggleButton

class **PictureToggleButton**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=110, NextFocusedWidget=0, MaxStringLength=32, AlternateFlag=False, AutomaticFocusMotion=False, PictureReference=1, AlternatePictureReference=1, PicturePosition='A661_LEFT', ToggleState='A661_UNSELECTED', Alignment='A661_LEFT', LabelString='LabelString', AlternateLabelString='AlternateLabelString'*) → PictureToggleButton

Method to set the parameter(s) of PictureToggleButton.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for PictureToggleButton.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for PictureToggleButton.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for PictureToggleButton.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for PictureToggleButton.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for PictureToggleButton.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for PictureToggleButton.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for PictureToggleButton.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for PictureToggleButton.

alternate_flag(*v_alternate_flag: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_ALTERNATE_FLAG for PictureToggleButton.

auto_focus_motion(*v_automatic_focus_motion: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for PictureToggleButton.

picture_reference(*v_picture_reference*) → Widget

Method to build runtime message A661_PICTURE_REFERENCE for PictureToggleButton.

altern_picture_reference(*v_alternate_picture_reference*) → Widget

Method to build runtime message A661_ALTERN_PICTURE_REFERENCE for PictureToggleButton.

picture_position(*v_picture_position: int* | *str* | *a661_align_lcrtb*) → Widget

Method to build runtime message A661_PICTURE_POSITION for PictureToggleButton.

inner_state_toggle(*v_toggle_state: int | str | a661_selection*) → Widget

Method to build runtime message A661_INNER_STATE_TOGGLE for PictureToggleButton.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for PictureToggleButton.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for PictureToggleButton.

string_alternate(*v_alternate_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING_ALTERNATE for PictureToggleButton.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for PictureToggleButton.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for PictureToggleButton.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for PictureToggleButton.

PopUpMenu

class **PopUpMenu**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*OpeningMode='A661_OPEN_UP', NumberOfEntries=4, PosX=0, PosY=0, SizeX=3000, SizeY=4900, MaxStringLength=32, StyleSet=110, PopUpIdentArray=[0, 0, 0, 0], PictureArray=[0, 0, 0, 0], EnableArray=[True, True, True, True], StringArray=['Entry1', 'Entry2', 'Entry3', 'Entry4']*) → PopUpMenu

Method to set the parameter(s) of PopUpMenu.

opening_mode(*v_opening_mode: int | str | a661_menu_opening_mode*) → Widget

Method to build runtime message A661_OPENING_MODE for PopUpMenu.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for PopUpMenu.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for PopUpMenu.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for PopUpMenu.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for PopUpMenu.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for PopUpMenu.

picture_entry_array(*v_array, v_number_of_entries=None*) → Widget

Method to build runtime message A661_PICTURE_ENTRY_ARRAY for PopUpMenu.

enable_entry_array(*v_array: List[Tuple[int, int | str | bool | a661_bool]], v_number_of_entries=None*) → Widget

Method to build runtime message A661_ENABLE_ENTRY_ARRAY for PopUpMenu.

string_array(*v_array, v_number_of_strings=None*) → Widget

Method to build runtime message A661_STRING_ARRAY for PopUpMenu.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for PopUpMenu.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for PopUpMenu.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for PopUpMenu.

entry_pop_up_array(*v_array, v_number_of_entries=None*) → Widget

Method to build runtime message A661_ENTRY_POP_UP_ARRAY for PopUpMenu.

PopUpMenuButton

class **PopUpMenuButton**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=110, NextFocusedWidget=0, PopupPosX=0, PopupPosY=700, PopupSizeX=3000, PopupSizeY=3000, MaxStringLength=32, MaxStringLengthPopUp=32, PictureReference=1, NumberOfEntries=4, PicturePosition='A661_RIGHT', OpeningMode='A661_OPEN_UP', Alignment='A661_LEFT', AutomaticFocusMotion=False, LabelString='LabelString', PopUpIdentArray=[0, 0, 0, 0], PictureArray=[0, 0, 0, 0], EnableArray=[True, True, True, True], StringArray=['Entry1', 'Entry2', 'Entry3', 'Entry4']*) → PopUpMenuButton

Method to set the parameter(s) of PopUpMenuButton.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for PopUpMenuButton.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for PopUpMenuButton.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for PopUpMenuButton.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for PopUpMenuButton.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for PopUpMenuButton.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for PopUpMenuButton.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for PopUpMenuButton.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for PopUpMenuButton.

popup_pos_x(*v_popup_pos_x*) → Widget

Method to build runtime message A661_POPUP_POS_X for PopUpMenuButton.

popup_pos_y(*v_popup_pos_y*) → Widget

Method to build runtime message A661_POPUP_POS_Y for PopUpMenuButton.

popup_size_x(*v_popup_size_x*) → Widget

Method to build runtime message A661_POPUP_SIZE_X for PopUpMenuButton.

popup_size_y(*v_popup_size_y*) → Widget

Method to build runtime message A661_POPUP_SIZE_Y for PopUpMenuButton.

picture_reference(*v_picture_reference*) → Widget

Method to build runtime message A661_PICTURE_REFERENCE for PopUpMenuButton.

picture_position(*v_picture_position: int* | *str* | *a661_align_lcrtb*) → Widget

Method to build runtime message A661_PICTURE_POSITION for PopUpMenuButton.

opening_mode(*v_opening_mode: int* | *str* | *a661_menu_opening_mode*) → Widget

Method to build runtime message A661_OPENING_MODE for PopUpMenuButton.

alignment(*v_alignment: int* | *str* | *a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for PopUpMenuButton.

auto_focus_motion(*v_automatic_focus_motion: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for PopUpMenuButton.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for PopUpMenuButton.

picture_entry_array(*v_array, v_number_of_entries_updated=None*) → Widget

Method to build runtime message A661_PICTURE_ENTRY_ARRAY for PopUpMenuButton.

enable_entry_array(*v_array: List[Tuple[int, int | str | bool | a661_bool]], v_number_of_entries_updated=None*) → Widget

Method to build runtime message A661_ENABLE_ENTRY_ARRAY for PopUpMenuButton.

string_array(*v_array, v_number_of_strings=None*) → Widget

Method to build runtime message A661_STRING_ARRAY for PopUpMenuButton.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for PopUpMenuButton.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for PopUpMenuButton.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for PopUpMenuButton.

entry_pop_up_array(*v_array, v_number_of_entries_updated=None*) → Widget

Method to build runtime message A661_ENTRY_POP_UP_ARRAY for PopUpMenuButton.

PopUpPanel

class **PopUpPanel**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*UAPositionFlag=True, AutomaticClosure=False, PosX=0, PosY=0, SizeX=3000, SizeY=3000, StyleSet=100*) → PopUpPanel

Method to set the parameter(s) of PopUpPanel.

ua_position_flag(*v_uaposition_flag: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_UA_POSITION_FLAG for PopUpPanel.

auto_closure(*v_automatic_closure: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_CLOSURE for PopUpPanel.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for PopUpPanel.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for PopUpPanel.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for PopUpPanel.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for PopUpPanel.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for PopUpPanel.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for PopUpPanel.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for PopUpPanel.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for PopUpPanel.

PopUpPanelButton

class **PopUpPanelButton**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=110, NextFocusedWidget=0, PopupPosX=0, PopupPosY=700, PopupSizeX=3000, PopupSizeY=3000, MaxStringLength=32, PictureReference=1, PicturePosition='A661_RIGHT', Alignment='A661_LEFT', UAPositionFlag=True, AutomaticClosure=False, LabelString='LabelString'*) → PopUpPanelButton

Method to set the parameter(s) of PopUpPanelButton.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for PopUpPanelButton.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for PopUpPanelButton.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for PopUpPanelButton.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for PopUpPanelButton.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for PopUpPanelButton.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for PopUpPanelButton.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for PopUpPanelButton.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for PopUpPanelButton.

popup_pos_x(*v_popup_pos_x*) → Widget

Method to build runtime message A661_POPUP_POS_X for PopUpPanelButton.

popup_pos_y(*v_popup_pos_y*) → Widget

Method to build runtime message A661_POPUP_POS_Y for PopUpPanelButton.

popup_size_x(*v_popup_size_x*) → Widget

Method to build runtime message A661_POPUP_SIZE_X for PopUpPanelButton.

popup_size_y(*v_popup_size_y*) → Widget

Method to build runtime message A661_POPUP_SIZE_Y for PopUpPanelButton.

picture_reference(*v_picture_reference*) → Widget

Method to build runtime message A661_PICTURE_REFERENCE for PopUpPanelButton.

picture_position(*v_picture_position: int* | *str* | *a661_align_lcrtb*) → Widget

Method to build runtime message A661_PICTURE_POSITION for PopUpPanelButton.

alignment(*v_alignment: int* | *str* | *a661_alignment*) → Widget

Method to PopUpPanelButton runtime message A661_ALIGNMENT for PopUpMenuButton.

ua_position_flag(*v_uaposition_flag: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_UA_POSITION_FLAG for PopUpPanelButton.

auto_closure(*v_automatic_closure: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_AUTO_CLOSURE for PopUpPanelButton.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for PopUpPanelButton.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for PopUpPanelButton.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for PopUpPanelButton.

ProxyButton

class **ProxyButton**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', DedicatedKeyIdent=0, TargetWidgetIdent=0*) → ProxyButton

Method to set the parameter(s) of ProxyButton.

target_widget_id(*v_target_widget_ident*) → Widget

Method to build runtime message A661_TARGET_WIDGET_ID for ProxyButton.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for ProxyButton.

PushButton

class **PushButton**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=110, NextFocusedWidget=0, MaxStringLength=32, AutomaticFocusMotion=False, Alignment='A661_LEFT', LabelString='LabelString'*) → PushButton

Method to set the parameter(s) of PushButton.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for PushButton.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for PushButton.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for PushButton.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for PushButton.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for PushButton.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for PushButton.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for PushButton.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for PushButton.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for PushButton.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to PopUpPanelButton runtime message A661_ALIGNMENT for PushButton.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for PushButton.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for PushButton.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for PushButton.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for PushButton.

RadioBox

class **RadioBox**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True*) → RadioBox

Method to set the parameter(s) of RadioBox.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for RadioBox.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for RadioBox.

RotationContainer

class **RotationContainer**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Visible=True, CenterX=0, CenterY=0, RotationAngle=0.0*) → RotationContainer

Method to set the parameter(s) of RotationContainer.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for RotationContainer.

center_xy(*v_center_x, v_center_y*) → Widget

Method to build runtime message A661_CENTER_XY for RotationContainer.

center_x(*v_center_x*) → Widget

Method to build runtime message A661_CENTER_X for RotationContainer.

center_y(*v_center_y*) → Widget

Method to build runtime message A661_CENTER_Y for RotationContainer.

rotation_angle(*v_rotation_angle*) → Widget

Method to build runtime message A661_ROTATION_ANGLE for RotationContainer.

ScrollList

class **ScrollList**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=4900, StyleSet=110, NextFocusedWidget=0, MaxNumberOfEntries=16, NumberOfEntries=4, SelectedEntry=1, MaxStringLength=32, FirstAccessibleEntry=1, FirstVisibleEntry=1, VerticalScroll='A661_RIGHT', Alignment='A661_LEFT', FlagReportVisibleEntry=False, AutomaticFocusMotion=False, DefaultStyleText=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], EnableArray=[True, True, True, True], LabelStringArray=['Entry1', 'Entry2', 'Entry3', 'Entry4']*) → ScrollList

Method to set the parameter(s) of ScrollList.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for ScrollList.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for ScrollList.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for ScrollList.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for ScrollList.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for ScrollList.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for ScrollList.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for ScrollList.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for ScrollList.

number_of_entries(*v_number_of_entries*) → Widget

Method to build runtime message A661_NUMBER_OF_ENTRIES for ScrollList.

selected_entry(*v_selected_entry*) → Widget

Method to build runtime message A661_SELECTED_ENTRY for ScrollList.

first_access_entry(*v_first_accessible_entry*) → Widget

Method to build runtime message A661_FIRST_ACCESS_ENTRY for ScrollList.

first_visible_entry(*v_first_visible_entry*) → Widget

Method to build runtime message A661_FIRST_VISIBLE_ENTRY for ScrollList.

vertical_scroll(*v_vertical_scroll: int | str | a661_align_lartb*) → Widget

Method to build runtime message A661_VERTICAL_SCROLL for ScrollList.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to PopUpPanelButton runtime message A661_ALIGNMENT for ScrollList.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for ScrollList.

enable_entry_array(*v_array: List[Tuple[int, int | str | bool | a661_bool]], v_number_of_entries_updated=None*) → Widget

Method to build runtime message A661_ENABLE_ENTRY_ARRAY for ScrollList.

string_array(*v_array, v_number_of_strings=None*) → Widget

Method to build runtime message A661_STRING_ARRAY for ScrollList.

shift_first_visible_entry(*v_shift_first_visible_entry*) → Widget

Method to build runtime message A661_SHIFT_FIRST_VISIBLE_ENTRY for ScrollList.

number_of_ua_entries(*v_number_of_uaentries*) → Widget

Method to build runtime message A661_NUMBER_OF_UA_ENTRIES for ScrollList.

first_access_ua_entry(*v_first_accessible_uaentry*) → Widget

Method to build runtime message A661_FIRST_ACCESS_UA_ENTRY for ScrollList.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for ScrollList.

pos_xy(*v_pos_x*, *v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for ScrollList.

size_xy(*v_size_x*, *v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for ScrollList.

entry_array(*v_array*, *v_number_of_entries_updated=None*) → Widget

Method to build runtime message A661_ENTRY_ARRAY for ScrollList.

ScrollPanel

class **ScrollPanel**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=4000, SizeY=4000, LineDeltaX=100, LineDeltaY=100, PageDeltaX=500, PageDeltaY=500, HomeX=0, HomeY=0, FrameX=200, FrameY=200, SizeXsheet=10000, SizeYsheet=10000, BoundX=0, BoundY=0, SizeXbound=10000, SizeYbound=10000, StyleSet=110, HorizontalScroll='A661_TOP', VerticalScroll='A661_RIGHT', FlagReportFramePos=False*) → ScrollPanel

Method to set the parameter(s) of ScrollPanel.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for ScrollPanel.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for ScrollPanel.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for ScrollPanel.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for ScrollPanel.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for ScrollPanel.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for ScrollPanel.

line_delta_x(*v_line_delta_x*)

Method to build runtime message A661_LINE_DELTA_X for ScrollPanel.

line_delta_y(*v_line_delta_y*) → Widget

Method to build runtime message A661_LINE_DELTA_Y for ScrollPanel.

page_delta_x(*v_page_delta_x*) → Widget

Method to build runtime message A661_PAGE_DELTA_X for ScrollPanel.

page_delta_y(*v_page_delta_y*) → Widget

Method to build runtime message A661_PAGE_DELTA_Y for ScrollPanel.

home_x(*v_home_x*) → Widget

Method to build runtime message A661_HOME_X for ScrollPanel.

home_y(*v_home_y*) → Widget

Method to build runtime message A661_HOME_Y for ScrollPanel.

frame_x(*v_frame_x*) → Widget

Method to build runtime message A661_FRAME_X for ScrollPanel.

frame_y(*v_frame_y*) → Widget

Method to build runtime message A661_FRAME_Y for ScrollPanel.

sheet_size_x(*v_size_xsheet*) → Widget

Method to build runtime message A661_SHEET_SIZE_X for ScrollPanel.

sheet_size_y(*v_size_ysheet*) → Widget

Method to build runtime message A661_SHEET_SIZE_Y for ScrollPanel.

bound_x(*v_bound_x*) → Widget

Method to build runtime message A661_BOUND_X for ScrollPanel.

bound_y(*v_bound_y*) → Widget

Method to build runtime message A661_BOUND_Y for ScrollPanel.

bound_size_x(*v_size_xbound*) → Widget

Method to build runtime message A661_BOUND_SIZE_X for ScrollPanel.

bound_size_y(*v_size_ybound*) → Widget

Method to build runtime message A661_BOUND_SIZE_Y for ScrollPanel.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for ScrollPanel.

horizontal_scroll(*v_horizontal_scroll: int | str | a661_align_lartb*) → Widget

Method to build runtime message A661_HORIZONTAL_SCROLL for ScrollPanel.

vertical_scroll(*v_vertical_scroll: int | str | a661_align_lartb*) → Widget

Method to build runtime message A661_VERTICAL_SCROLL for ScrollPanel.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for ScrollPanel.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for ScrollPanel.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for ScrollPanel.

frame_xy(*v_frame_x, v_frame_y*) → Widget

Method to build runtime message A661_FRAME_XY for ScrollPanel.

bound_xy(*v_bound_x, v_bound_y*) → Widget

Method to build runtime message A661_BOUND_XY for ScrollPanel.

bound_size_xy(*v_size_xbound, v_size_ybound*) → Widget

Method to build runtime message A661_BOUND_SIZE_XY for ScrollPanel.

ScrollWheelArea

class **ScrollWheelArea**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', PosX=0, PosY=0, SizeX=2000, SizeY=2000*) → ScrollWheelArea

Method to set the parameter(s) of ScrollWheelArea.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for ScrollWheelArea.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for ScrollWheelArea.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for ScrollWheelArea.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for ScrollWheelArea.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for ScrollWheelArea.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for ScrollWheelArea.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for ScrollWheelArea.

SelectionListButton

class **SelectionListButton**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, SelectingAreaWidth=3000, SelectingAreaHeight=4900, StyleSet=110, NextFocusedWidget=0, MaxNumberOfEntries=16, NumberOfEntries=4, SelectedEntry=1, MaxStringLength=32, OpeningEntry=1, Alignment='A661_LEFT', OpeningMode='A661_OPEN_DOWN', AutomaticFocusMotion=False, LabelString='Selection', EntryList=['Entry1', 'Entry2', 'Entry3', 'Entry4']*) → SelectionListButton

Method to set the parameter(s) of SelectionListButton.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for SelectionListButton.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for SelectionListButton.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for SelectionListButton.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for SelectionListButton.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for SelectionListButton.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for SelectionListButton.

selecting_width(*v_selecting_area_width*) → Widget

Method to build runtime message A661_SELECTING_WIDTH for SelectionListButton.

selecting_height(*v_selecting_area_height*) → Widget

Method to build runtime message A661_SELECTING_HEIGHT for SelectionListButton.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for SelectionListButton.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for SelectionListButton.

number_of_entries(*v_number_of_entries*) → Widget

Method to build runtime message A661_NUMBER_OF_ENTRIES for SelectionListButton.

selected_entry(*v_selected_entry*) → Widget

Method to build runtime message A661_SELECTED_ENTRY for SelectionListButton.

opening_entry(*v_opening_entry*) → Widget

Method to build runtime message A661_OPENING_ENTRY for SelectionListButton.

alignment(*v_alignment: int* | *str* | *a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for SelectionListButton.

opening_mode(*v_opening_mode: int* | *str* | *a661_menu_opening_mode*) → Widget

Method to build runtime message A661_OPENING_MODE for SelectionListButton.

auto_focus_motion(*v_automatic_focus_motion: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for SelectionListButton.

string_array(*v_array, v_number_of_strings=None*) → Widget

Method to build runtime message A661_STRING_ARRAY for SelectionListButton.

entry_valid(*v_entry_validation: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for SelectionListButton.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for SelectionListButton.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for SelectionListButton.

ShuffleToFitContainer

class **ShuffleToFitContainer**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, NumberOfVisibleChildren=3, ShuffleToFitMode='A661_SHUFFLE_UP', ItemSpacing=100, StyleSet=1*) → ShuffleToFitContainer

Method to set the parameter(s) of ShuffleToFitContainer.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for ShuffleToFitContainer.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for ShuffleToFitContainer.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for ShuffleToFitContainer.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for ShuffleToFitContainer.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for ShuffleToFitContainer.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for ShuffleToFitContainer.

number_of_visible_children(*v_number_of_visible_children*) → Widget

Method to build runtime message A661_NUMBER_OF_VISIBLE_CHILDREN for ShuffleToFitContainer.

shuffle_to_fit_mode(*v_shuffle_to_fit_mode: int | str | a661_shuffle_mode*) → Widget

Method to build runtime message A661_SHUFFLE_TO_FIT_MODE for ShuffleToFitContainer.

item_spacing(*v_item_spacing*) → Widget

Method to build runtime message A661_ITEM_SPACING for ShuffleToFitContainer.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for ShuffleToFitContainer.

pos_xy(*v_pos_x*, *v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for ShuffleToFitContainer.

size_xy(*v_size_x*, *v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for ShuffleToFitContainer.

SizeToFitContainer

class **SizeToFitContainer**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, NumberOfVisibleChildren=3, SizeToFitMode='A661_SIZE_TOP_DOWN', ItemSpacing=100*) → SizeToFitContainer

Method to set the parameter(s) of SizeToFitContainer.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for SizeToFitContainer.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for SizeToFitContainer.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for SizeToFitContainer.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for SizeToFitContainer.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for SizeToFitContainer.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for SizeToFitContainer.

number_of_visible_children(*v_number_of_visible_children*) → Widget

Method to build runtime message A661_NUMBER_OF_VISIBLE_CHILDREN for SizeToFitContainer.

size_to_fit_mode(*v_size_to_fit_mode: int | str | a661_size_mode*) → Widget

Method to build runtime message A661_SIZE_TO_FIT_MODE for SizeToFitContainer.

item_spacing(*v_item_spacing*) → Widget

Method to build runtime message A661_ITEM_SPACING for SizeToFitContainer.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for SizeToFitContainer.

pos_xy(*v_pos_x*, *v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for SizeToFitContainer.

size_xy(*v_size_x*, *v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for SizeToFitContainer.

Slider

class **Slider**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=1, NextFocusedWidget=0, MinValue=0.0, MaxValue=100.0, Value=0.0, MajorTickInterval=20.0, ShowMajorLabels=True, Orientation='A661_LEFT_TO_RIGHT', Alignment='A661_LEFT', AutomaticFocusMotion=False, LsbMultiple=5, MinorTickMultiple=4, MajorTickReference=0.0*) → Slider

Method to set the parameter(s) of Slider.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for Slider.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for Slider.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for Slider.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for Slider.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for Slider.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for Slider.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for Slider.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for Slider.

min_value(*v_min_value*) → Widget

Method to build runtime message A661_MIN_VALUE for Slider.

max_value(*v_max_value*) → Widget

Method to build runtime message A661_MAX_VALUE for Slider.

value(*v_value*) → Widget

Method to build runtime message A661_VALUE for Slider.

major_tick_interval(*v_major_tick_interval*) → Widget

Method to build runtime message A661_MAJOR_TICK_INTERVAL for Slider.

show_major_labels(*v_show_major_labels: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_SHOW_MAJOR_LABELS for Slider.

slider_orientation(*v_orientation: int* | *str* | *a661_orientation*) → Widget

Method to build runtime message A661_SLIDER_ORIENTATION for Slider.

alignment(*v_alignment: int* | *str* | *a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for Slider.

auto_focus_motion(*v_automatic_focus_motion: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for Slider.

lsb_multiple(*v_lsb_multiple*) → Widget

Method to build runtime message A661_LSB_MULTIPLE for Slider.

minor_tick_multiple(*v_minor_tick_multiple*) → Widget

Method to build runtime message A661_MINOR_TICK_MULTIPLE for Slider.

major_tick_reference(*v_major_tick_reference*) → Widget

Method to build runtime message A661_MAJOR_TICK_REFERENCE for Slider.

entry_valid(*v_entry_validation: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for Slider.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for Slider.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for Slider.

Symbol

class **Symbol**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*MotionAllowed=True, Visible=True, PosX=0, PosY=0, RotationAngle=0.0, StyleSet=1, SymbolReference=1, ColorIndex=1*) → Symbol

Method to set the parameter(s) of Symbol.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for Symbol.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for Symbol.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for Symbol.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for Symbol.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for Symbol.

orientation(*v_rotation_angle*) → Widget

Method to build runtime message A661_ORIENTATION for Symbol.

color_index(*v_color_index*) → Widget

Method to build runtime message A661_COLOR_INDEX for Symbol.

symbol_reference(*v_symbol_reference*) → Widget

Method to build runtime message A661_SYMBOL_REFERENCE for Symbol.

SymbolAnimated

class **SymbolAnimated**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*MotionAllowed=True, Visible=True, PosX=0, PosY=0, StyleSet=1, ColorIndex=1, LoopType='A661_LOOP_FORWARD', NumberOfSymbols=3, IndexOfFrequency=10, AnimationType='A661_RUN', SymbolArray=[1, 2, 3], OrientationArray=[0.0, 0.0, 0.0], MovementXArray=[0, 0, 0], MovementYArray=[0, 0, 0]*) → SymbolAnimated

Method to set the parameter(s) of SymbolAnimated.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for SymbolAnimated.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for SymbolAnimated.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for SymbolAnimated.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for SymbolAnimated.

color_index(*v_color_index*) → Widget

Method to build runtime message A661_COLOR_INDEX for SymbolAnimated.

loop_flag(*v_loop_flag: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_LOOP_FLAG for SymbolAnimated.

index_of_frequency(*v_index_of_frequency*) → Widget

Method to build message A661_INDEX_OF_FREQUENCY for SymbolAnimated.

animation_type(*v_animation_type: int | str | a661_animation_type*) → Widget

Method to build runtime message A661_ANIMATION_TYPE for SymbolAnimated.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for SymbolAnimated.

SymbolPushButton

class **SymbolPushButton**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=110, NextFocusedWidget=0, SymbolReference=1, MaxStringLength=32, SymbolPosition='A661_RIGHT', AutomaticFocusMotion=False, Alignment='A661_LEFT', LabelString='LabelString'*) → SymbolPushButton

Method to set the parameter(s) of SymbolPushButton.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for SymbolPushButton.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for SymbolPushButton.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for SymbolPushButton.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for SymbolPushButton.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for SymbolPushButton.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for SymbolPushButton.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for SymbolPushButton.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for SymbolPushButton.

symbol_reference(*v_symbol_reference*) → Widget

Method to build runtime message A661_SYMBOL_REFERENCE for SymbolPushButton.

symbol_position(*v_symbol_position: int* | *str* | *a661_align_lcrtb*) → Widget

Method to build runtime message A661_SYMBOL_POSITION for SymbolPushButton.

auto_focus_motion(*v_automatic_focus_motion: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for SymbolPushButton.

alignment(*v_alignment: int* | *str* | *a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for SymbolPushButton.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for SymbolPushButton.

entry_valid(*v_entry_validation: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for SymbolPushButton.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for SymbolPushButton.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for SymbolPushButton.

SymbolToggleButton

class **SymbolToggleButton**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=110, NextFocusedWidget=0, MaxStringLength=32, AlternateFlag=False, AutomaticFocusMotion=False, SymbolReference=1, AlternateSymbolReference=1, SymbolPosition='A661_LEFT', ToggleState='A661_UNSELECTED', Alignment='A661_LEFT', LabelString='LabelString', AlternateLabelString='AlternateLabelString'*) → SymbolToggleButton

Method to set the parameter(s) of SymbolToggleButton.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for SymbolToggleButton.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for SymbolToggleButton.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for SymbolToggleButton.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for SymbolToggleButton.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for SymbolToggleButton.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for SymbolToggleButton.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for SymbolToggleButton.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for SymbolToggleButton.

alternate_flag(*v_alternate_flag: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ALTERNATE_FLAG for SymbolToggleButton.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for SymbolToggleButton.

symbol_reference(*v_symbol_reference*) → Widget

Method to build runtime message A661_SYMBOL_REFERENCE for SymbolToggleButton.

altern_symbol_reference(*v_alternate_symbol_reference*) → Widget

Method to build runtime message A661_ALTERN_SYMBOL_REFERENCE for SymbolToggleButton.

symbol_position(*v_symbol_position: int | str | a661_align_lcrtb*) → Widget

Method to build runtime message A661_SYMBOL_POSITION for SymbolToggleButton.

inner_state_toggle(*v_toggle_state: int | str | a661_selection*) → Widget

Method to build runtime message A661_INNER_STATE_TOGGLE for SymbolToggleButton.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for SymbolToggleButton.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for SymbolToggleButton.

string_alternate(*v_alternate_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING_ALTERNATE for SymbolToggleButton.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for SymbolToggleButton.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for SymbolToggleButton.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for SymbolToggleButton.

TabbedPanel

class **TabbedPanel**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, StyleSet=100, NextFocusedWidget=0, MaxStringLength=32, PictureReference=1, PicturePosition='A661_CENTER', AutomaticFocusMotion=False, Alignment='A661_LEFT', InsetSize=1000, LabelString='LabelString'*) → TabbedPanel

Method to set the parameter(s) of TabbedPanel.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for TabbedPanel.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for TabbedPanel.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for TabbedPanel.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for TabbedPanel.

picture_reference(*v_picture_reference*) → Widget

Method to build runtime message A661_PICTURE_REFERENCE for TabbedPanel.

picture_position(*v_picture_position: int | str | a661_align_lcrtb*) → Widget

Method to build runtime message A661_PICTURE_POSITION for TabbedPanel.

auto_focus_motion(*v_automatic_focus_motion: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for TabbedPanel.

alignment(*v_alignment: int | str | a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for TabbedPanel.

inset_size(*v_inset_size*) → Widget

Method to build runtime message A661_INSET_SIZE for TabbedPanel.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for TabbedPanel.

TabbedPanelGroup

class **TabbedPanelGroup**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=3000, StyleSet=0, ActiveTabbedPanelID=0, TabPosition='A661_TOP', AutomaticInsetSizeFlag=True*) → TabbedPanelGroup

Method to set the parameter(s) of TabbedPanelGroup.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for TabbedPanelGroup.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for TabbedPanelGroup.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for TabbedPanelGroup.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for TabbedPanelGroup.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for TabbedPanelGroup.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for TabbedPanelGroup.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for TabbedPanelGroup.

active_tabbed_panel(*v_active_tabbed_panel_id*) → Widget

Method to build runtime message A661_ACTIVE_TABBED_PANEL for TabbedPanelGroup.

tab_position(**v_tab_position: int | str | a661_align_lartb**) → Widget

Method to build runtime message A661_TAB_POSITION for TabbedPanelGroup.

entry_valid(*v_entry_validation: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for TabbedPanelGroup.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for TabbedPanelGroup.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for TabbedPanelGroup.

ToggleButton

class **ToggleButton**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, PosX=0, PosY=0, SizeX=3000, SizeY=800, StyleSet=110, NextFocusedWidget=0, MaxStringLength=32, ToggleState='A661_UNSELECTED', AlternateFlag=False, AutomaticFocusMotion=False, Alignment='A661_LEFT', LabelString='LabelString', AlternateLabelString='AlternateLabelString'*) → ToggleButton

Method to set the parameter(s) of ToggleButton.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for ToggleButton.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for ToggleButton.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for ToggleButton.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for ToggleButton.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for ToggleButton.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for ToggleButton.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for ToggleButton.

next_focused_widget(*v_next_focused_widget*) → Widget

Method to build runtime message A661_NEXT_FOCUSED_WIDGET for ToggleButton.

inner_state_toggle(*v_toggle_state: int | str | a661_selection*) → Widget

Method to build runtime message A661_INNER_STATE_TOGGLE for ToggleButton.

alternate_flag(*v_alternate_flag: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_ALTERNATE_FLAG for ToggleButton.

auto_focus_motion(*v_automatic_focus_motion: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_AUTO_FOCUS_MOTION for ToggleButton.

alignment(*v_alignment: int* | *str* | *a661_alignment*) → Widget

Method to build runtime message A661_ALIGNMENT for ToggleButton.

string(*v_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING for ToggleButton.

string_alternate(*v_alternate_label_string, v_string_size=None*) → Widget

Method to build runtime message A661_STRING_ALTERNATE for ToggleButton.

entry_valid(*v_entry_validation: int* | *str* | *bool* | *a661_bool*) → Widget

Method to build runtime message A661_ENTRY_VALID for ToggleButton.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for ToggleButton.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for ToggleButton.

TouchArea

class **TouchArea**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Enable='A661_TRUE', Visible=True, StyleSet=0, PosX=0, PosY=0, SizeX=2000, SizeY=2000, ReportMode='A661_REPORT_ON_TRANSITION', AllowOutside=False, MaxTouchPoints=10, NumberOfTouchPoints=5*) → TouchArea

Method to set the parameter(s) of TouchArea.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for TouchArea.

enable(*v_enable: int | str | a661_bool_with_validation*) → Widget

Method to build runtime message A661_ENABLE for TouchArea.

style_set(*v_style_set*) → Widget

Method to build runtime message A661_STYLE_SET for TouchArea.

pos_x(*v_pos_x*) → Widget

Method to build runtime message A661_POS_X for TouchArea.

pos_y(*v_pos_y*) → Widget

Method to build runtime message A661_POS_Y for TouchArea.

pos_xy(*v_pos_x, v_pos_y*) → Widget

Method to build runtime message A661_POS_XY for TouchArea.

size_x(*v_size_x*) → Widget

Method to build runtime message A661_SIZE_X for TouchArea.

size_y(*v_size_y*) → Widget

Method to build runtime message A661_SIZE_Y for TouchArea.

size_xy(*v_size_x, v_size_y*) → Widget

Method to build runtime message A661_SIZE_XY for TouchArea.

number_of_touch_points(*v_number_of_touch_points*) → Widget

Method to build runtime message A661_NUMBER_OF_TOUCH_POINTS for TouchArea.

TranslationContainer

class **TranslationContainer**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*Visible=True, TranslationX=0, TranslationY=0*) → TranslationContainer

Method to set the parameter(s) of TranslationContainer.

visible(*v_visible: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_VISIBLE for TranslationContainer.

translation_xy(*v_translation_x, v_translation_y*) → Widget

Method to build runtime message A661_TRANSLATION_XY for TranslationContainer.

translation_x(*v_translation_x*) → Widget

Method to build runtime message A661_TRANSLATION_X for TranslationContainer.

translation_y(*v_translation_y*) → Widget

Method to build runtime message A661_TRANSLATION_Y for TranslationContainer.

WatchdogContainer

class **WatchdogContainer**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

define(*TimePeriod=10, ValidCountLimit=100, FailCountLimit=80, Refresh=0, ShowIfFailIdent=0, ShowFail=False*) → WatchdogContainer

Method to set the parameter(s) of WatchdogContainer.

refresh(*v_refresh*) → Widget

Method to build runtime message A661_REFRESH for WatchdogContainer.

show_fail(*v_show_fail: int | str | bool | a661_bool*) → Widget

Method to build runtime message A661_SHOW_FAIL for WatchdogContainer.

Widget Extensions

All widget extension classes inherit from "ExtensionWidget Class".

- "CursorEntryEventsExtension"
- "CursorEventsExtension"
- "CursorShapeExtension"
- "DirectionalTabbingExtension"
- "FocusStopExtension"
- "InitialFocusExtension"
- "LegendStringAlignmentExtension"
- "MultiSelectionExtension"
- "PictureExtension"
- "SymbolExtension"
- "TicsArrayExtension"
- "VisibleChildIndexExtension"
- "WordWrapExtension"

CursorEntryEventsExtension

class **CursorEntryEventsExtension**(*parent_id: int = 0*)

define(*NumEvents=5, EventIdArray=['A661_CURSOR_PRESSED', 'A661_CURSOR_RELEASED', 'A661_CURSOR_CLICKED', 'A661_CURSOR_DOUBLE_CLICKED', 'A661_CURSOR_RIGHT_CLICKED']*) → CursorEntryEventsExtension

Method to set the parameter(s) of extension CursorEntryEventsExtension.

CursorEventsExtension

class **CursorEventsExtension**(*parent_id: int = 0*)

define(*NumEvents=5, EventIdArray=['A661_CURSOR_PRESSED', 'A661_CURSOR_RELEASED', 'A661_CURSOR_CLICKED', 'A661_CURSOR_DOUBLE_CLICKED', 'A661_CURSOR_RIGHT_CLICKED']*) → CursorEventsExtension

Method to set the parameter(s) of extension CursorEventsExtension.

CursorShapeExtension

class **CursorShapeExtension**(*parent_id: int = 0*)

define(AllowChange=True, CursorIndex=5) → CursorShapeExtension

Method to set the parameter(s) of extension CursorShapeExtension.

allow_change(*v_allow_change: int | str | bool | a661_bool*) → ExtensionWidget

Method to build runtime message A661_ALLOW_CHANGE for extension CursorShapeExtension.

cursor_index(*v_cursor_index*) → ExtensionWidget

Method to build runtime message A661_CURSOR_INDEX for extension CursorShapeExtension.

DirectionalTabbingExtension

class **DirectionalTabbingExtension**(parent_id: int = 0)

define(*NumDirections=4, TabDirectionWidgetIdArray=[0, 0, 0, 0]*) → DirectionalTabbingExtension

Method to set the parameter(s) of extension DirectionalTabbingExtension.

tab_direction_widget_id_array(*v_array, v_number_of_entries_updated=None*) → ExtensionWidget

Method to build runtime message A661_TAB_DIRECTION_WIDGET_ID_ARRAY for extension DirectionalTabbingExtension.

FocusStopExtension

class **FocusStopExtension**(*parent_id: int = 0*)

define(*StopForward=False, StopBackward=False*) → FocusStopExtension

Method to set the parameter(s) of extension FocusStopExtension.

stop_forward(*v_stop_forward: int | str | bool | a661_bool*) → ExtensionWidget

Method to build runtime message A661_STOP_FORWARD for extension FocusStopExtension.

stop_backward(*v_stop_backward: int | str | bool | a661_bool*) → ExtensionWidget

Method to build runtime message A661_STOP_BACKWARD for extension FocusStopExtension.

InitialFocusExtension

class **InitialFocusExtension**(*parent_id: int = 0*)

define(*InitialFocus=False*) → InitialFocusExtension

Method to set the parameter(s) of extension InitialFocusExtension.

initial_focus(*v_initial_focus: int | str | bool | a661_bool*) → ExtensionWidget

Method to build runtime message A661_INITIAL_FOCUS for extension InitialFocusExtension.

LegendStringAlignmentExtension

class **LegendStringAlignmentExtension**(*parent_id: int = 0*)

define(*LegendAlignment='A661_LEFT'*) → LegendStringAlignmentExtension

Method to set the parameter(s) of extension LegendStringAlignmentExtension.

MultiSelectionExtension

class **MultiSelectionExtension**(*parent_id: int = 0*)

define(MaximumSelectableEntries=16) → MultiSelectionExtension

Method to set the parameter(s) of extension MultiSelectionExtension.

select_all(*v_select_all*) → ExtensionWidget

Method to build runtime message A661_SELECT_ALL for extension MultiSelectionExtension.

unselect_all(*v_unselect_all*) → ExtensionWidget

Method to build runtime message A661_UNSELECT_ALL for extension MultiSelectionExtension.

selected_entry_array(*v_array, v_number_of_entries_updated=None*) → ExtensionWidget

Method to build runtime message A661_SELECTED_ENTRY_ARRAY for extension MultiSelectionExtension.

PictureExtension

class **PictureExtension**(*parent_id: int = 0*)

define(MaxNumberOfPictures=16, NumberOfPictures=1, PicturePosition='A661_RIGHT', PictureArray=[1]) → PictureExtension

Method to set the parameter(s) of extension PictureExtension.

number_of_pictures(*v_number_of_pictures*) → ExtensionWidget

Method to build runtime message A661_NUMBER_OF_PICTURES for extension PictureExtension.

picture_entry_array(*v_array, v_number_of_entries_updated=None*) → ExtensionWidget

Method to build runtime message A661_PICTURE_ENTRY_ARRAY for extension PictureExtension.

SymbolExtension

class **SymbolExtension**(*parent_id: int = 0*)

define(MaxNumberOfSymbols=16, NumberOfSymbols=1, SymbolPosition='A661_RIGHT', SymbolArray=[1]) → SymbolExtension

Method to set the parameter(s) of extension SymbolExtension.

number_of_symbols(*v_number_of_symbols*) → ExtensionWidget

Method to build runtime message A661_NUMBER_OF_SYMBOLS for extension SymbolExtension.

symbol_entry_array(*v_array, v_number_of_entries_updated=None*) → ExtensionWidget

Method to build runtime message A661_SYMBOL_ENTRY_ARRAY for extension SymbolExtension.

TicsArrayExtension

class **TicsArrayExtension**(*parent_id: int = 0*)

define(*MaxNumberOfTicsCoarse=16, NumberOfTicsCoarse=3, MaxNumberOfTicsFine=16, NumberOfTicsFine=3, TicsCoarseArray=[{'Tics': 5.0, 'ValueMax': 10.0}, {'Tics': 15.0, 'ValueMax': 30.0}, {'Tics': 25.0, 'ValueMax': 50.0}], TicsFineArray=[{'Tics': 0.3, 'ValueMax': 10.0}, {'Tics': 0.5, 'ValueMax': 30.0}, {'Tics': 0.7, 'ValueMax': 50.0}]*) → TicsArrayExtension

Method to set the parameter(s) of extension TicsArrayExtension.

number_of_tics_coarse(*v_number_of_tics_coarse*) → ExtensionWidget

Method to build runtime message A661_NUMBER_OF_TICS_COARSE for extension TicsArrayExtension.

number_of_tics_fine(*v_number_of_tics_fine*) → ExtensionWidget

Method to build runtime message A661_NUMBER_OF_TICS_FINE for extension TicsArrayExtension.

tics_coarse_array(*v_array*, *v_number_of_entries_updated=None*) → ExtensionWidget

Method to build runtime message A661_TICS_COARSE_ARRAY for extension TicsArrayExtension.

tics_fine_array(*v_array*, *v_number_of_entries_updated=None*) → ExtensionWidget

Method to build runtime message A661_TICS_FINE_ARRAY for extension TicsArrayExtension.

VisibleChildIndexExtension

class **VisibleChildIndexExtension**(*parent_id: int = 0*)

define(VisibleChildIndex=0) → VisibleChildIndexExtension

Method to set the parameter(s) of extension VisibleChildIndexExtension.

visible_child_index(*v_visible_child_index*) → ExtensionWidget

Method to build runtime message A661_VISIBLE_CHILD_INDEX for extension VisibleChildIndexExtension.

WordWrapExtension

class **WordWrapExtension**(*parent_id: int = 0*)

define(*WordWrap=False, WrapStyle='A661_PRESERVE_WORD'*) → WordWrapExtension

Method to set the parameter(s) of extension WordWrapExtension.

word_wrap(*v_word_wrap: int* | *str* | *bool* | *a661_bool*) → ExtensionWidget

Method to build runtime message A661_WORD_WRAP for extension WordWrapExtension.

wrap_style(*v_wrap_style: int* | *str* | *a661_wrap_style*) → ExtensionWidget

Method to build runtime message A661_WRAP_STYLE for extension WordWrapExtension.

Symbol Commands

ArcCircle

class **ArcCircle**(*children: List[SymbolCmd] | None = None*)

define(*Filled=False, PosX=0, PosY=0, Radius=2000, StartAngle=0.0, EndAngle=90.0*) → ArcCircle

Method to set the parameter(s) of symbol command ArcCircle.

ArcEllipse

class **ArcEllipse**(*children: List[SymbolCmd] | None = None*)

define(*Filled=False, PosX=0, PosY=0, SizeX=2000, SizeY=600, StartAngle=0.0, EndAngle=90.0*) → ArcEllipse

Method to set the parameter(s) of symbol command ArcEllipse.

BoundingCircle

class **BoundingCircle**(*children: List[SymbolCmd] | None = None*)

define(Radius=2000) → BoundingCircle

Method to set the parameter(s) of symbol command BoundingCircle.

BoundingRectangle

class **BoundingRectangle**(*children: List[SymbolCmd] | None = None*)

define(*OffsetX=0, OffsetY=0, SizeX=4000, SizeY=4000*) → BoundingRectangle

Method to set the parameter(s) of symbol command BoundingRectangle.

CircularSensitiveArea

class **CircularSensitiveArea**(*children: List[SymbolCmd] | None = None*)

define(*PosX=0, PosY=0, Radius=2000*) → CircularSensitiveArea

Method to set the parameter(s) of symbol command CircularSensitiveArea.

Crown

class **Crown**(*children: List[SymbolCmd] | None = None*)

define(*Filled=False, PosX=0, PosY=0, InnerRadius=1500, OuterRadius=2000, StartAngle=0.0, EndAngle=90.0*) → Crown

Method to set the parameter(s) of symbol command Crown.

EndGroup

class **EndGroup**(*children: List[SymbolCmd] | None = None*)

define() → EndGroup

No parameter for symbol command EndGroup (use of method define is not required).

Focus

class **Focus**(*children: List[SymbolCmd] | None = None*)

define() → Focus

No parameter for symbol command Focus (use of method define is not required).

Highlight

class **Highlight**(*children: List[SymbolCmd] | None = None*)

define() → Highlight

No parameter for symbol command Highlight (use of method define is not required).

LegendAnchor

class **LegendAnchor**(*children: List[SymbolCmd]* | *None = None*)

define(*AssociatedLegendType='A661_LEGEND_ONLY', PosX=0, PosY=0*) → LegendAnchor

Method to set the parameter(s) of symbol command LegendAnchor.

Line

class **Line**(*children: List[SymbolCmd]* | *None = None*)

define(PosXStart=0, PosYStart=0, PosXEnd=1000, PosYEnd=1000) → Line

Method to set the parameter(s) of symbol command Line.

LinePolar

class **LinePolar**(*children: List[SymbolCmd]* | *None = None*)

define(*PosXStart=0, PosYStart=0, RotationAngle=45.0, LineLength=1000*) → LinePolar

Method to set the parameter(s) of symbol command LinePolar.

Polyline

class **Polyline**(*children: List[SymbolCmd]* | *None = None*)

define(*NumVertices=3, Closed=False, Vertices=[{'x': 0, 'y': 0}, {'x': 1000, 'y': 1000}, {'x': 2000, 'y': 1000}]*) → Polyline

Method to set the parameter(s) of symbol command Polyline.

Rectangle

class **Rectangle**(*children: List[SymbolCmd]* | *None = None*)

define(*Filled=False, PosX=0, PosY=0, SizeX=2000, SizeY=600*) → Rectangle

Method to set the parameter(s) of symbol command Rectangle.

RectangularSensitiveArea

class **RectangularSensitiveArea**(*children: List[SymbolCmd]* | *None* = *None*)

define(*RotationAllowed=False, PosX=-1000, PosY=-1000, SizeX=2000, SizeY=2000*) → RectangularSensitiveArea

Method to set the parameter(s) of symbol command RectangularSensitiveArea.

SetColor

class **SetColor**(*children: List[SymbolCmd]* | *None* = *None*)

define(ColorIndex=1) → SetColor

Method to set the parameter(s) of symbol command SetColor.

SetFont

class **SetFont**(*children: List[SymbolCmd]* | *None* = *None*)

define(*Font=0*) → SetFont

Method to set the parameter(s) of symbol command SetFont.

SetHalo

class **SetHalo**(*children: List[SymbolCmd]* | *None* = *None*)

define(*Halo='A661_FALSE'*) → SetHalo

Method to set the parameter(s) of symbol command SetHalo.

SetLineStyle

class **SetLineStyle**(*children: List[SymbolCmd]* | *None* = *None*)

define(*Style=0*) → SetLineStyle

Method to set the parameter(s) of symbol command SetLineStyle.

SetRotation

class **SetRotation**(*children: List[SymbolCmd]* | *None* = *None*)

define(RotationAngle=0.0, CenterX=0, CenterY=0) → SetRotation

Method to set the parameter(s) of symbol command SetRotation.

SetTranslation

class **SetTranslation**(*children: List[SymbolCmd]* | *None = None*)

define(*TranslationX=0, TranslationY=0*) → SetTranslation

Method to set the parameter(s) of symbol command SetTranslation.

Standard

class **Standard**(*children: List[SymbolCmd]* | *None = None*)

define() → Standard

No parameter for symbol command Standard (use of method define is not required).

StartGroup

class **StartGroup**(*children: List[SymbolCmd]* | *None = None*)

define() → StartGroup

No parameter for symbol command StartGroup (use of method define is not required).

Text

class **Text**(*children: List[SymbolCmd]* | *None = None*)

define(*Alignment='A661_CENTER', PosX=0, PosY=0, RotationAngle=0.0, StringText='Text'*) → Text

Method to set the parameter(s) of symbol command Text.

Triangle

class **Triangle**(*children: List[SymbolCmd]* | *None = None*)

define(Filled=False, PosX=0, PosY=0, PosX2=600, PosY2=1000, PosX3=1200, PosY3=0) → Triangle

Method to set the parameter(s) of symbol command Triangle.

TriangleFan

class **TriangleFan**(*children: List[SymbolCmd]* | *None = None*)

define(*NumVertices=3, Vertices=[{'x': 0, 'y': 0}, {'x': 1000, 'y': 1000}, {'x': 2000, 'y': 1000}]*) → TriangleFan

Method to set the parameter(s) of symbol command TriangleFan.

TriangleStrip

class **TriangleStrip**(*children: List[SymbolCmd]* | *None = None*)

define(*NumVertices=3, Vertices=[{'x': 0, 'y': 0}, {'x': 1000, 'y': 1000}, {'x': 2000, 'y': 1000}]*) → TriangleStrip

Method to set the parameter(s) of symbol command TriangleStrip.

Map Items

ItemListBuffer

class **ItemListBuffer**(*map_vert: bool = False, data_format: str | bool | int | IntEnum | None = None, data_format_y: str | bool | int | IntEnum | None = None*)

Bases: object

Item list for Map (MapHorz_ItemList and MapVert_ItemList widgets).

Arguments

map_vert: Set to True to indicate that the items are for MapVert (default is MapHorz).

data_format: Gives the data format used for the coordinates. When not provided, long 'raw' value is expected.

- For MapHorz, it applies for X and Y coordinates and corresponds to the MapDataFormat parameter of the MapHorz_Source widget (enum *a661_MapDataFormat_type*, uchar or string corresponding to the A661 constant).

  Possible values are: A661_MDF_BRG_DIST_ACHDG, A661_MDF_DIST_DIST, A661_MDF_DIST_DIST_FIX and A661_MDF_LAT_LONG.
- For MapVert, it applies only for the X coordinate and corresponds to the MapDataFormatX parameter of the MapVert_Source widget (enum *a661_MapDataFormatX_type*, uchar or string corresponding to the A661 constant).

  Possible values are: A661_MDF_ABSOLUTE, A661_MDF_RELATIVE and A661_MDF_X_DIST.

data_format_y: Gives the data format used for the Y coordinate for MapVert (ignored for MapHorz). When not provided, long 'raw' value is expected.

It corresponds to the MapDataFormatY parameter of the MapVert_Source widget (enum *a661_MapDataFormatY_type*, uchar or string corresponding to the A661 constant).

Possible values are: A661_MDF_ABSOLUTE, A661_MDF_RELATIVE and A661_MDF_Y_ALT.

Items are added to ItemListBuffer object using methods add_<item> where item is the name of the map item in lower case.

Example:

```
ItemListBuffer().add_item_style(3).add_symbol_generic(True,1,False,1
000,1000)
```

By default, the item index is calculated automatically (the index is increased each time the method is called on the object). It can however be explicitly set for each of the items, as last argument of the corresponding method.

empty() → ItemListBuffer

Remove all items from the ItemListBuffer object and reset the item index.

get_current_index() → int

Get current item index.

add_item_raw(*msg: Msg*) → ItemListBuffer

Add item from Msg object.

Arguments

msg: Msg object containing the item (starting by the item index).

add_draw_line_to_cursor(*relative_position: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, index: int = -1*) → ItemListBuffer

Draw_Line_To_Cursor (A661_DRAW_LINE_TO_CURSOR).

Arguments

relative_position: When set to True (A661_TRUE), X and Y are screen unit values relative to last symbol.

x_coord: X coordinate (depends on data_format and relative_position).

y_coord: Y coordinate (depends on data_format(_y) and relative_position).

index: Optional item index.

add_filled_poly_start(*fill_style_index: int, x_coord: int | float, y_coord: int | float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Filled_Poly_Start (A661_FILLED_POLY_START),

Filled_Poly_Start_Interactive (A661_FILLED_POLY_START_INTERACTIVE).

Arguments

fill_style_index: Color of the polygon.

x_coord: X coordinate of the polygon start point (depends on data_format).

y_coord: Y coordinate of the polygon start point (depends on map_data_format(_y)).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_filled_poly_start_interactive(*fill_style_index: int, x_coord: int | float, y_coord: int | float, index: int = -1*) → ItemListBuffer

Filled_Poly_Start_Interactive (A661_FILLED_POLY_START_INTERACTIVE).

Interactive version of Symbol_Filled_Poly_Start.

Same as add_filled_poly_start *with interactive flag set to True*.

add_item_style(*item_style_set: int, index: int = -1*) → ItemListBuffer

Item_Style (A661_ITEM_STYLE).

Arguments

item_style_set: Item style (based on the styleset capacities of the Map ItemList widget).

index: Optional item index.

add_item_synchronization(*data_type: int, synchronization_data_1: int, synchronization_data_2: int, index: int = -1*) → ItemListBuffer

Item_Synchronization (A661_ITEM_SYNCHRONIZATION).

Arguments

data_type: synchronization data type (uchar).

synchronization_data_1: synchronization data (ulong).

synchronization_data_2: synchronization data (ulong).

index: Optional item index.

add_legend(*end_flag: str | bool | int | IntEnum, legend_string: str, index: int = -1*) → ItemListBuffer

Legend (A661_LEGEND).

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item is the last of the legend group.

legend_string: Legend string (16 bytes maximum).

index: Optional item index.

add_legend_combo(end_flag: str | bool | int | IntEnum, legend_types: str | int | UserEnum, legend_string: str, index: int = -1) → ItemListBuffer

Legend_Combo (A661_LEGEND_COMBO).

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item is the last of the legend group.

legend_types: Type of legend (enum *a661_legend_type*, uchar or string corresponding to the A661 constant).

Possible values are: A661_LEGEND_ONLY, A661_POPUP_ONLY, A661_HIGHLIGHT_ONLY, A661_LEGEND_AND_POPUP, A661_LEGEND_AND_HIGHLIGHT, A661_POPUP_AND_HIGHLIGHT and A661_LEGEND_AND_POPUP_AND_HIGHLIGHT.

legend_string: Legend string (16 bytes maximum).

index: Optional item index.

add_legend_highlight(*end_flag: str | bool | int | IntEnum, legend_string: str, index: int = -1*) → ItemListBuffer

Legend_Highlight (A661_LEGEND_HIGHLIGHT).

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item is the last of the legend group.

legend_string: Legend string (16 bytes maximum).

index: Optional item index.

add_legend_pop_up(*end_flag: str | bool | int | IntEnum, legend_string: str, index: int = -1*) → ItemListBuffer

Legend_Pop_Up (A661_LEGEND_POP_UP).

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item is the last of the legend group.

legend_string: Legend string (16 bytes maximum).

index: Optional item index.

add_legend_anchor(*relative_position: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, index: int = -1*) → ItemListBuffer

Legend_Anchor (A661_LEGEND_ANCHOR).

Arguments

relative_position: When set to True (A661_TRUE), X and Y are screen unit values relative to last symbol.

x_coord: X coordinate (depends on data_format and relative_position).

y_coord: Y coordinate (depends on data_format(_y) and relative_position).

index: Optional item index.

add_legend_anchor_rotated(*relative_position: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, orientation: float, legend_flip: str | bool | int | IntEnum, index: int = -1*) → ItemListBuffer

Legend_Anchor_Rotated (A661_LEGEND_ANCHOR_ROTATED).

Arguments

relative_position: When set to True (A661_TRUE), X and Y are screen unit values relative to last symbol.

x_coord: X coordinate (depends on data_format and relative_position).

y_coord: Y coordinate (depends on data_format(_y) and relative_position).

orientation: Legend orientation (-180.0 to 180.0 degrees) (positive is clockwise).

- For MapHorz, the orientation is relative to True North.
- For MapVert, the orientation is relative to upright axis of the MapVert display.

legend_flip: When set to True (A661_TRUE), the legend is flipped over according to orientation.

index: Optional item index.

add_line_start(*x_coord: int | float, y_coord: int | float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Line_Start (A661_LINE_START),
Line_Start_Interactive (A661_LINE_START_INTERACTIVE).

Arguments

x_coord: X coordinate (depends on data_format).

y_coord: Y coordinate (depends on data_format(_y)).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_line_start_interactive(*x_coord: int | float, y_coord: int | float, index: int = -1*) → ItemListBuffer

Line_Start_Interactive (A661_LINE_START_INTERACTIVE).
Interactive version of Line_Start.
Same as add_line_start *with interactive flag set to True*.

add_line_segment(*end_flag: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Line_Segment (A661_LINE_SEGMENT),

Line_Segment_Interactive (A661_LINE_SEGMENT_INTERACTIVE).

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item is the last of the line or polygon group.

x_coord: X coordinate of the segment (depends on data_format).

y_coord: Y coordinate of the segment (depends on data_format(_y)).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_line_segment_interactive(*end_flag: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, index: int = -1*) → ItemListBuffer

Line_Segment_Interactive (A661_LINE_SEGMENT_INTERACTIVE).

Interactive version of Line_Segment.

Same as add_line_segment *with interactive flag set to True*.

add_line_arc(*end_flag: str | bool | int | IntEnum, inbound_course: float, radius: float, course_change: float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Line_Arc (A661_LINE_ARC),

Line_Arc_Interactive (A661_LINE_ARC_INTERACTIVE).

Only for MapHorz.

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item is the last of the line group.

inbound_course: inbound course angle (-180.0 to 180.0 degrees).

radius: radius in nautical miles (-32768.0 to 32768.0) (negative value for left turn).

course_change: course change angle (-180.0 to 180.0 degrees).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_line_arc_interactive(*end_flag: str | bool | int | IntEnum, inbound_course: float, radius: float, course_change: float, index: int = -1*) → ItemListBuffer

Line_Arc_Interactive (A661_LINE_ARC_INTERACTIVE).

Interactive version of Line_Arc.

Same as add_line_arc *with interactive flag set to True*.

add_not_used(*index: int = -1*) → ItemListBuffer

Not_Used (A661_NOT_USED).

Arguments

index: Optional item index.

add_symbol_generic(*end_flag: str | bool | int | IntEnum, symbol_type: int, relative_position: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Generic (A661_SYMBOL_GENERIC),

Symbol_Generic_Interactive (A661_SYMBOL_GENERIC_INTERACTIVE).

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item does not have legend.

symbol_type: Symbol type.

relative_position: When set to True (A661_TRUE), X and Y are screen unit values relative to last symbol.

x_coord: X coordinate (depends on data_format and relative_position).

y_coord: Y coordinate (depends on data_format(_y) and relative_position).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_symbol_generic_interactive(*end_flag: str | bool | int | IntEnum, symbol_type: int, relative_position: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, index: int = -1*) → ItemListBuffer

Symbol_Generic_Interactive (A661_SYMBOL_GENERIC_INTERACTIVE).

Interactive version of Symbol_Generic.

Same as add_symbol_generic *with interactive flag set to True*.

add_symbol_rotated(*end_flag: str | bool | int | IntEnum, symbol_type: int, relative_position: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, orientation: float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Rotated (A661_SYMBOL_ROTATED),

Symbol_Rotated_Interactive (A661_SYMBOL_ROTATED_INTERACTIVE).

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item does not have legend.

symbol_type: Symbol type.

relative_position: When set to True (A661_TRUE), X and Y are screen unit values relative to last symbol.

x_coord: X coordinate (depends on data_format and relative_position).

y_coord: Y coordinate (depends on data_format(_y) and relative_position).

orientation: Symbol orientation (-180.0 to 180.0 degrees).

- For MapHorz, the orientation is relative to True North (positive is clockwise).
- For MapVert, the orientation is relative to upright axis of the MapVert display (positive is counter-clockwise).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_symbol_rotated_interactive(*end_flag: str | bool | int | IntEnum, symbol_type: int, relative_position: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, orientation: float, index: int = -1*) → ItemListBuffer

Symbol_Rotated_Interactive (A661_SYMBOL_ROTATED_INTERACTIVE).

Interactive version of Symbol_Rotated.

Same as add_symbol_rotated *with interactive flag set to True*.

add_symbol_target(*end_flag: str | bool | int | IntEnum, symbol_type: int, length: int, x_coord: int | float, y_coord: int | float, orientation: float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Target (A661_SYMBOL_TARGET),

Symbol_Target_Interactive (A661_SYMBOL_TARGET_INTERACTIVE).

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item does not have legend.

symbol_type: Symbol type.

length: Velocity/Distance/Altitude illustrated by variable length part of the symbol (16bits integer value representing Knots/Nautical Miles/Feet).

x_coord: X coordinate (depends on data_format).

y_coord: Y coordinate (depends on data_format(_y)).

orientation: Symbol orientation (-180.0 to 180.0 degrees).

- For MapHorz, the orientation is relative to True North (positive is clockwise).
- For MapVert, the orientation is relative to upright axis of the MapVert display (positive is counter-clockwise).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_symbol_target_interactive(*end_flag: str | bool | int | IntEnum, symbol_type: int, length: int, x_coord: int | float, y_coord: int | float, orientation: float, index: int = -1*) → ItemListBuffer

Symbol_Target_Interactive (A661_SYMBOL_TARGET_INTERACTIVE).

Interactive version of Symbol_Target.

Same as add_symbol_target *with interactive flag set to True*.

add_symbol_runway(*end_flag: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, length: float, orientation: float = 0.0, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Runway (A661_SYMBOL_RUNWAY),

Symbol_Runway_Interactive (A661_SYMBOL_RUNWAY_INTERACTIVE).

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item does not have legend.

x_coord: X coordinate (depends on data_format).

y_coord: Y coordinate (depends on data_format(_y)).

length: Length of the runway in feet (up to 32768.0 feet).

orientation (MapHorz only): Symbol orientation relative to True North (-180.0 to 180.0 degrees) (positive is clockwise).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_symbol_runway_interactive(*end_flag: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, length: float, orientation: float = 0.0, index: int = -1*) → ItemListBuffer

Symbol_Runway_Interactive (A661_SYMBOL_RUNWAY_INTERACTIVE).

Interactive version of Symbol_Runway.

Same as add_symbol_runway *with interactive flag set to True*.

add_symbol_circle(*end_flag: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, radius: float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Circle (A661_SYMBOL_CIRCLE),

Symbol_Circle_Interactive (A661_SYMBOL_CIRCLE_INTERACTIVE).

Only for MapHorz.

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item does not have legend.

x_coord: X coordinate (depends on data_format).

y_coord: Y coordinate (depends on data_format(_y)).

radius: Radius in nautical miles (up to 32768.0 miles).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_symbol_circle_interactive(*end_flag: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, radius: float, index: int = -1*) → ItemListBuffer

Symbol_Circle_Interactive (A661_SYMBOL_CIRCLE_INTERACTIVE).

Interactive version of Symbol_Circle.

Same as add_symbol_circle *with interactive flag set to True*.

add_symbol_oval(*fill_style_index: int, x_coord: int | float, y_coord: int | float, radius: float, axis_ratio: float, orientation: float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Oval (A661_SYMBOL_OVAL),
Symbol_Oval_Interactive (A661_SYMBOL_OVAL_INTERACTIVE).
Only for MapHorz.

Arguments

fill_style_index: Color of the oval.
x_coord: X coordinate (depends on data_format).
y_coord: Y coordinate (depends on data_format(_y)).
radius: Radius (Half the Major Axis) in nautical miles (up to 32768.0 miles).
axis_ratio: Side ratio (Minor Axis divided by Major Axis) (greater than 0.0 and up to 1.0).
orientation: Major Axis orientation relative to True North (-180.0 to 180.0 degrees) (positive is clockwise).
interactive: Set to True to use interactive version of item (False by default).
index: Optional item index.

add_symbol_oval_interactive(*fill_style_index: int, x_coord: int | float, y_coord: int | float, radius: float, axis_ratio: float, orientation: float, index: int = -1*) → ItemListBuffer

Symbol_Oval_Interactive (A661_SYMBOL_OVAL_INTERACTIVE).
Interactive version of Symbol_Oval.
Same as add_symbol_oval *with interactive flag set to True*.

add_symbol_rectangle(*fill_style_index: int, x_coord: int | float, y_coord: int | float, major_size_length: float, axis_ratio: float, orientation: float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Rectangle (A661_SYMBOL_RECTANGLE),

Symbol_Rectangle_Interactive (A661_SYMBOL_RECTANGLE_INTERACTIVE).

Arguments

fill_style_index: Color of the rectangle.

x_coord: X coordinate of the rectangle reference point (depends on data_format).

y_coord: Y coordinate of the rectangle reference point (depends on data_format(_y)).

major_size_length: Length of the rectangle major side in nautical miles (up to 32768.0 miles).

axis_ratio: Side ratio (Minor Side divided by Major Side) (greater than 0.0 and up to 1.0).

orientation: Major Side orientation (-180.0 to 180.0 degrees).

- For MapHorz, the orientation is relative to True North (positive is clockwise).
- For MapVert, the orientation is relative to upright axis of the MapVert display (positive is counter-clockwise).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_symbol_rectangle_interactive(*fill_style_index: int, x_coord: int | float, y_coord: int | float, major_size_length: float, axis_ratio: float, orientation: float, index: int = -1*) → ItemListBuffer

Symbol_Rectangle_Interactive (A661_SYMBOL_RECTANGLE_INTERACTIVE).

Interactive version of Symbol_Rectangle.

Same as add_symbol_rectangle *with interactive flag set to True*.

add_symbol_parkable(*end_flag: str | bool | int | IntEnum, symbol_type: int, parked_symbol_type: int, relative_position: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Parkable (A661_SYMBOL_PARKABLE),

Symbol_Parkable_Interactive (A661_SYMBOL_PARKABLE_INTERACTIVE).

Only for MapHorz.

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item does not have legend.

symbol_type: Symbol type when the symbol is not parked.

parked_symbol_type: Symbol type when the symbol is parked.

relative_position: When set to True (A661_TRUE), X and Y are screen unit values relative to last symbol.

x_coord: X coordinate (depends on data_format and relative_position).

y_coord: Y coordinate (depends on data_format(_y) and relative_position).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_symbol_parkable_interactive(*end_flag: str | bool | int | IntEnum, symbol_type: int, parked_symbol_type: int, relative_position: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, index: int = -1*) → ItemListBuffer

Symbol_Parked_Interactive (A661_SYMBOL_PARKED_INTERACTIVE).

Interactive version of Symbol_Parked.

Same as add_symbol_parked *with interactive flag set to True*.

add_symbol_rotated_parkable(*end_flag: str | bool | int | IntEnum, symbol_type: int, parked_symbol_type: int, relative_position: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, orientation: float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Rotated_Parkable (A661_SYMBOL_ROTATED_PARKABLE), Symbol_Rotated_Interactive_Parkable (A661_SYMBOL_ROTATED_PARKABLE_INTERACTIVE).

Only for MapHorz.

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item does not have legend.

symbol_type: Symbol type when the symbol is not parked.

parked_symbol_type: Symbol type when the symbol is parked.

relative_position: When set to True (A661_TRUE), X and Y are screen unit values relative to last symbol.

x_coord: X coordinate (depends on data_format and relative_position).

y_coord: Y coordinate (depends on data_format(_y) and relative_position).

orientation: Symbol orientation (-180.0 to 180.0 degrees), relative to True North (positive is clockwise).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_symbol_rotated_parkable_interactive(*end_flag: str | bool | int | IntEnum, symbol_type: int, parked_symbol_type: int, relative_position: str | bool | int | IntEnum, x_coord: int | float, y_coord: int | float, orientation: float, index: int = -1*) → ItemListBuffer

Symbol_Rotated_Parkable_Interactive (A661_SYMBOL_ROTATED_PARKABLE_INTERACTIVE).

Interactive version of Symbol_Rotated_Parkable.

Same as add_symbol_rotated_parkable *with interactive flag set to True*.

add_symbol_target_parkable(*end_flag: str | bool | int | IntEnum, symbol_type: int, parked_symbol_type: int, relative_position: str | bool | int | IntEnum, length: int, x_coord: int | float, y_coord: int | float, orientation: float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Target_Parkable (A661_SYMBOL_TARGET_PARKABLE), Symbol_Target_Parkable_Interactive (A661_SYMBOL_TARGET_PARKABLE_INTERACTIVE).

Only for MapHorz.

Arguments

end_flag: Set to True (A661_TRUE) to indicate that the item does not have legend.

symbol_type: Symbol type when the symbol is not parked.

parked_symbol_type: Symbol type when the symbol is parked.

relative_position: When set to True (A661_TRUE), X and Y are screen unit values relative to last symbol.

length: Velocity/Distance/Altitude illustrated by variable length part of the symbol (16bits integer value representing Knots/Nautical Miles/Feet).

x_coord: X coordinate (depends on data_format).

y_coord: Y coordinate (depends on data_format(_y)).

orientation: Symbol orientation (-180.0 to 180.0 degrees), relative to True North (positive is clockwise).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_symbol_target_parkable_interactive(*end_flag: str | bool | int | IntEnum, symbol_type: int, parked_symbol_type: int, relative_position: str | bool | int | IntEnum, length: int, x_coord: int | float, y_coord: int | float, orientation: float, index: int = -1*) → ItemListBuffer

Symbol_Target_Parkable_Interactive (A661_SYMBOL_TARGET_PARKABLE_INTERACTIVE).

Interactive version of Symbol_Target_Parkable.

Same as add_symbol_target_parkable *with interactive flag set to True*.

add_parking_line_start(*x1_coord: int | float, y1_coord: int | float, orientation: float, symbol_type: int, parked_symbol_type: int, index: int = -1*) → ItemListBuffer

Parking_Line_Start (A661_PARKING_LINE_START).

Only for MapHorz.

Arguments

x1_coord: X coordinate of the start of the line (depends on data_format).

y1_coord: Y coordinate of the start of the line (depends on data_format(_y)).

orientation: Start and end line symbols orientation (-180.0 to 180.0 degrees), relative to True North (positive is clockwise).

symbol_type: Start line symbol type when the symbol is not parked.

parked_symbol_type: Start line symbol type when the symbol is parked.

index: Optional item index.

add_parking_line_end(*x2_coord: int | float, y2_coord: int | float, symbol_type: int, parked_symbol_type: int, index: int = -1*) → ItemListBuffer

Parking_Line_End (A661_PARKING_LINE_END).

Only for MapHorz.

Arguments

x2_coord: X coordinate of the end of the line (depends on data_format).

y2_coord: Y coordinate of the end of the line (depends on data_format(_y)).

symbol_type: End line symbol type when the symbol is not parked.

parked_symbol_type: End line symbol type when the symbol is parked.

index: Optional item index.

add_triangle_fan_start(*x1_coord: int | float, y1_coord: int | float, x2_coord: int | float, y2_coord: int | float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Triangle_Fan_Start (A661_TRIANGLE_FAN_START), Symbol_Triangle_Fan_Start_Interactive (A661_TRIANGLE_FAN_START_INTERACTIVE).

Arguments

x1_coord: X coordinate of first point (depends on data_format).

y1_coord: Y coordinate of first point (depends on data_format(_y)).

x2_coord: X coordinate of second point (depends on data_format).

y2_coord: Y coordinate of second point (depends on data_format(_y)).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_triangle_fan_start_interactive(*x1_coord: int | float, y1_coord: int | float, x2_coord: int | float, y2_coord: int | float, index: int = -1*) → ItemListBuffer

Symbol_Triangle_Fan_Start_Interactive (A661_TRIANGLE_FAN_START_INTERACTIVE).

Interactive version of Symbol_Triangle_Fan_Start.

Same as add_triangle_fan_start *with interactive flag set to True*.

add_triangle_strip_start(*x1_coord: int | float, y1_coord: int | float, x2_coord: int | float, y2_coord: int | float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Triangle_Strip_Start (A661_TRIANGLE_STRIP_START), Symbol_Triangle_Strip_Start_Interactive (A661_TRIANGLE_STRIP_START_INTERACTIVE).

Arguments

x1_coord: X coordinate of first point (depends on data_format).

y1_coord: Y coordinate of first point (depends on data_format(_y)).

x2_coord: X coordinate of second point (depends on data_format).

y2_coord: Y coordinate of second point (depends on data_format(_y)).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_triangle_strip_start_interactive(*x1_coord: int | float, y1_coord: int | float, x2_coord: int | float, y2_coord: int | float, index: int = -1*) → ItemListBuffer

Symbol_Triangle_Strip_Start_Interactive (A661_TRIANGLE_STRIP_START_INTERACTIVE).

Interactive version of Symbol_Triangle_Strip_Start.

Same as add_triangle_strip_start *with interactive flag set to True*.

add_triangle_segment(*fill_style_index: int, x_coord: int | float, y_coord: int | float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Triangle_Segment (A661_TRIANGLE_SEGMENT),

Symbol_Triangle_Segment_Interactive (A661_TRIANGLE_SEGMENT_INTERACTIVE).

Arguments

fill_style_index: Color of the triangle completed by this item.

x_coord: X coordinate of item point (depends on data_format).

y_coord: Y coordinate of item point (depends on data_format(_y)).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_triangle_segment_interactive(*fill_style_index: int, x_coord: int | float, y_coord: int | float, index: int = -1*) → ItemListBuffer

Symbol_Triangle_Segment_Interactive (A661_TRIANGLE_SEGMENT_INTERACTIVE).

Interactive version of Symbol_Triangle_Segment.

Same as add_triangle_segment *with interactive flag set to True*.

add_triangle_segment_double(*fill_style_index: int, x1_coord: int | float, y1_coord: int | float, x2_coord: int | float, y2_coord: int | float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Triangle_Segment_Double (A661_TRIANGLE_SEGMENT_DOUBLE), Symbol_Triangle_Segment_Double_Interactive (A661_TRIANGLE_SEGMENT_DOUBLE_INTERACTIVE).

Arguments

fill_style_index: Color of the triangles completed by this item.

x1_coord: X coordinate of item first point (depends on data_format).

y1_coord: Y coordinate of item first point (depends on data_format(_y)).

x2_coord: X coordinate of item second point (depends on data_format).

y2_coord: Y coordinate of item second point (depends on data_format(_y)).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_triangle_segment_double_interactive(*fill_style_index: int, x1_coord: int | float, y1_coord: int | float, x2_coord: int | float, y2_coord: int | float, index: int = -1*) → ItemListBuffer

Symbol_Triangle_Segment_Double_Interactive (A661_TRIANGLE_SEGMENT_DOUBLE_INTERACTIVE).

Interactive version of Symbol_Triangle_Segment_Double.

Same as add_triangle_segment_double *with interactive flag set to True*.

add_triangle_end(*fill_style_index: int, x_coord: int | float, y_coord: int | float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Triangle_End (A661_TRIANGLE_END),
Symbol_Triangle_End_Interactive (A661_TRIANGLE_END_INTERACTIVE).

Arguments

fill_style_index: Color of the triangle completed by this item.
x_coord: X coordinate of item point (depends on data_format).
y_coord: Y coordinate of item point (depends on data_format(_y)).
interactive: Set to True to use interactive version of item (False by default).
index: Optional item index.

add_triangle_end_interactive(*fill_style_index: int, x_coord: int | float, y_coord: int | float, index: int = -1*) → ItemListBuffer

Symbol_Triangle_End_Interactive (A661_TRIANGLE_END_INTERACTIVE).
Interactive version of Symbol_Triangle_End.
Same as add_triangle_end *with interactive flag set to True*.

add_triangle_end_double(*fill_style_index: int, x1_coord: int | float, y1_coord: int | float, x2_coord: int | float, y2_coord: int | float, interactive: bool = False, index: int = -1*) → ItemListBuffer

Symbol_Triangle_End_Double (A661_TRIANGLE_END_DOUBLE),
Symbol_Triangle_End_Double_Interactive (A661_TRIANGLE_END_DOUBLE_INTERACTIVE).

Arguments

fill_style_index: Color of the triangles completed by this item.

x1_coord: X coordinate of item first point (depends on data_format).

y1_coord: Y coordinate of item first point (depends on data_format(_y)).

x2_coord: X coordinate of item second point (depends on data_format).

y2_coord: Y coordinate of item second point (depends on data_format(_y)).

interactive: Set to True to use interactive version of item (False by default).

index: Optional item index.

add_triangle_end_double_interactive(*fill_style_index: int, x1_coord: int | float, y1_coord: int | float, x2_coord: int | float, y2_coord: int | float, index: int = -1*) → ItemListBuffer

Symbol_Triangle_End_Double_Interactive (A661_TRIANGLE_END_DOUBLE_INTERACTIVE).

Interactive version of Symbol_Triangle_End_Double.

Same as add_triangle_end_double *with interactive flag set to True*.

add_vertex_render_start(*primitive_type: str | int | UserEnum, vertex_buffer_ident: int, render_sequential: str | bool | int | IntEnum, num_indices: int, i1: int, i2: int, i3: int = 0, i4: int = 0, i5: int = 0, i6: int = 0, index: int = -1*) → ItemListBuffer

Vertex_Render_Start (A661_VERTEX_RENDER_START).

Only for MapHorz.

Arguments

primitive_type: Type of primitive (enum a661_vertex_primitive_type, int or string corresponding to the A661 constant).

Possible values are: A661_VERTEX_RENDER_LINES, A661_VERTEX_RENDER_LINE_LOOP, A661_VERTEX_RENDER_POLYLINE, A661_VERTEX_RENDER_TRIANGLE, A661_VERTEX_RENDER_TRIANGLE_FAN and A661_VERTEX_RENDER_TRIANGLE_STRIP.

vertex_buffer_ident: WidgetIdent of the MapHorz_VertexBuffer widget containing the vertex data.

render_sequential: Set to True (A661_TRUE) to indicate that the vertex is rendered sequentially. In this case, only the first 2 indices are used to define the range.

num_indices: Number of indices in this item (up to 6).

i1: First MapHorz_VertexBuffer index number or range start.

i2: Second MapHorz_VertexBuffer index number or range end.

i3 to i6: Optional Third to Sixth MapHorz_VertexBuffer index numbers (set to 0 when not used).

index: Optional item index.

add_vertex_render_segment(*indices: List[int], index: int = -1*) → ItemListBuffer

Vertex_Render_Segment (A661_VERTEX_RENDER_SEGMENT).

Only for MapHorz.

Arguments

indices: List of MapHorz_VertexBuffer index numbers (up to 8).

index: Optional item index.

add_vertex_render_end(*num_indices: int, indices: List[int], index: int = -1*) → ItemListBuffer

Vertex_Render_End (A661_VERTEX_RENDER_END).

Only for MapHorz.

Arguments

num_indices: Number of indices used in this item (up to 8).

indices: List of MapHorz_VertexBuffer index numbers (up to 8).

index: Optional item index.

Server Control Proxy

class **ServerControlProxy**

Bases: **`object`**

Server Control Proxy class

Arguments:

None

Once the connection with the server established, the associated UA can communicate with the server.

- The UA sends its commands using A661 Layer Request of type OEM_REQ_SERVER_CONTROL_PROXY on layer #0.
- For each command, the server sends back an A661 Layer Event Notification of type OEM_NOTE_SERVER_CONTROL_PROXY (from layer #0 of proxy UA).
- In case of error, an A661 Exception is sent (for the proxy UA).

class **Commands***(value)*

Bases: **`IntEnum`**

Commands identifier

INIT = *0*
CHANGE_ACTIVE_CURSOR = *1*
CHANGE_DISPLAY_CONF = *2*
GET_CURSOR_POINTER_DATA = *3*
GET_FOCUSED_WIDGET = *4*
GET_INTERFACE_DATA = *5*
GET_PARAMETER = *6*
GET_POPUP_STACK = *7*
GET_SCREENSHOT = *8*
GET_WIDGETS_UNDER_CURSOR = *9*
GET_WIDGETS_UNDER_GESTURE = *10*
OBSERVE_CDS_EVENTS = *11*
PUSH_BLOCK = *12*
PUSH_CURSOR = *13*
PUSH_GESTURE = *14*
PUSH_KEYBOARD = *15*
PUSH_TOUCH = *16*
SERVER_EXIT = *17*
SERVER_START = *18*
SET_CURSOR_ACTIVE_STATE = *19*
SET_INTERFACE_VALUE = *20*
STEP = *21*
COVERAGE_RESET = *32*
COVERAGE_DUMP = *33*
RUN_INTERACTIVE_SESSION = *48*

class **INTF_kind***(value)*

Bases: **`IntEnum`**

Interface kind (used to identify interface in notification returned after get_interface_data request)

SENSITIVEAREA = *0*
INTERACTIVITY = *1*
FOCUS = *2*
TABBEDPANEL = *3*
TOUCH = *4*

class **PARAM_kind***(value)*

Bases: **`IntEnum`**

Parameter type (used to identify type of data in notification returned after get_parameter request)

UNSIGNED = *0*

SIGNED = *1*

ENUM = *2*

FR = *3*

FLOAT = *4*

STRING = *5*

OTHER = *6*

MULTI = *7*

NOT_SUPPORTED = *8*

NOT_APPLICABLE = *9*

connect(*ua_id: int = 0, host: str = 'localhost', port: int = 1230, udp: bool = False*) → ServerControlProxy

Establishes the connection with the Server Control Proxy.

Arguments:

ua_id: Server Control Proxy UA identifier

host: server hostname (default is “localhost”)

port: socket base port (default is 1230)

udp: UDP/IP protocol when True, TCP/IP protocol when False (default is False)

disconnect() → ServerControlProxy

Ends the conection with the Server Control Proxy.

Arguments:

None

get_notifications() → List[Dict[str, Any]]

Gets pending notifications sent by the Server Control Proxy.

These notifications are either exceptions or Layer events of type OEM_NOTE_SERVER_CONTROL_PROXY.

cmd_raw(*cmd: int, data: Msg*) → None

Sends command to the Server Control Proxy.

Arguments:

cmd: command identifier

data: command data (Msg object)

cmd_change_active_cursor(*cursor: int = 0*) → None

Changes active cursor pointer and run the server for one step.

Arguments:

cursor: New active cursor number (from 1 to the number of cursor pointers declared in the server conf). 0 means select next cursor.

Command identifier: CHANGE_ACTIVE_CURSOR

cmd_change_display_conf(*conf: int = 0*) → None

Changes display configuration.

Arguments:

conf: New display configuration number (from 1 to the number of display configurations). 0 means select next conf.

Command identifier: CHANGE_DISPLAY_CONF

cmd_get_cursor_pointer_data(*cursor_id: int*) → None

Gets data of cursor pointer for the given pointer identifier.

Arguments:

cursor_id: cursor identifier

Command identifier: GET_CURSOR_POINTER_DATA

cmd_get_focused_widget(*app_id: int = 1, layer_id: int = 1*) → None

Gets the widget ID of the focused widget for the given user application/layer.

Arguments:

app_id: user application (UA) identifier

layer_id: layer identifier

Command identifier: GET_FOCUSED_WIDGET

cmd_get_interface_data(*interface_name: str, window_index: int = 0, app_id: int = 0, layer_id: int = 0, widget_id: int = 0*) → None

Gets the data of a given interface.

The supported interfaces are:

- Focus (data at window and widget levels),
- Interactivity (data at window and widget levels),
- SensitiveArea (data at window and widget level),
- TabbedPanel (data at widget level),
- Touch (data at server and window levels).

Arguments:

interface_name: interface name

window_index: display window index (when applicable)

app_id: user application identifier (UA) (0 if not used)

layer_id: layer identifier (applicable when app_id is not equal to 0)

widget_id: widget identifier (applicable when app_id is not equal to 0)

Command identifier: GET_INTERFACE_DATA

cmd_get_parameter(*app_id: int, layer_id: int, widget_id: int, param_name: str, first_elem: int = 0, nb_elems: int = 0*) → None

Gets the type and the value for the widget parameter.

Arguments:

app_id: user application (UA) identifier

layer_id: layer identifier

widget_id: widget identifier

param_name: widget parameter name

first_elem: index of first element to get in case of array parameter

nb_elems: number of elements to get in case of array parameter (0 means all elements)

Command identifier: GET_PARAMETER

cmd_get_popup_stack(*app_id: int, layer_id: int*) → None

Gets the list of widget ID which popup are currently opened.

Arguments:

app_id: user application (UA) identifier

layer_id: layer identifier

Command identifier: GET_POPUP_STACK

cmd_get_screenshot(*x: int = 0, y: int = 0, width: int = 0, height: int = 0, filename: str = 'screenshot', du_id: int = 1*) → None

Gets a screenshot of the Display Unit.

Arguments:

x: X origin position in pixels

y: Y origin position in pixels

width: width size of screenshot in pixels

height: height size of screenshot in pixels

filename: path to the saved screenshot raw file (.raw added to file name).

du_id: DU identifier

Command identifier: GET_SCREENSHOT

cmd_get_widgets_under_cursor(*app_id: int, layer_id: int*) → None

Gets the list of widget ID having the cursor.

Arguments:

layer_id: layer identifier

app_id: user application (UA) identifier

Command identifier: GET_WIDGETS_UNDER_CURSOR

cmd_get_widgets_under_gesture(*app_id: int, layer_id: int, type_gesture: int*) → None

Gets the list of widget ID having a given type of gesture.

Arguments:

app_id: user application (UA) identifier

layer_id: layer identifier

type_gesture: gesture type (enum Server.GestureType)

Command identifier: GET_WIDGETS_UNDER_GESTURE

cmd_observe_cds_events(*notify: bool*) → None

Observes CDS events (input device events and A661 notifications) while the server is running (i.e. it executes cycle steps).

Arguments:

notify: When set to True, the server will send, at each executed cycle step, OEM_NOTE_SERVER_CONTROL_PROXY notifications of type OBSERVE_CDS_EVENTS when for this cycle device events or A661 notifications are produced.

Command identifier: OBSERVE_CDS_EVENTS

cmd_push_block(*app_id: int*) → None

Notifies the Server Control Proxy that an A661 block has been sent to the server.

Arguments:

app_id: User application (UA) identifier for which the A661 block has been sent.

Command identifier: PUSH_BLOCK

cmd_push_gesture(*type_gesture: int, number_points: int = 1, state: int = GestureState.START, pos_x: int = 0, pos_y: int = 0, bounding_x1: int = 0, bounding_y1: int = 0, bounding_x2: int = 0, bounding_y2: int = 0, add_value1: float = 0.0, add_value2: float = 0.0, device_id: int = 1, du_id: int = 1*) → None

Sends a gesture device event to the server.

Arguments:

type_gesture: gesture type (enum *Server.GestureType*)

number_points: number of touch points

state: gesture state (enum Server.GestureState). For stateless gestures (TAP, DOUBLETAP, TAPCONFIRMED and FLICK), the value is ignored (always set to NONE)

pos_x: X position of gesture in user units

pos_y: Y position of gesture in user units

bounding_x1: X first position of bounding box (not required if only one touch point)

bounding_y1: Y first position of bounding box (not required if only one touch point)

bounding_x2: X second position of bounding box (not required if only one touch point)

bounding_y2: Y second position of bounding box (not required if only one touch point)

add_value1: first additional value depending on gesture type

add_value2: second additional value depending on gesture type

device_id: gesture device identifier

du_id: DU identifier

Command identifier: PUSH_GESTURE

cmd_push_keyboard(*key_code: int, modifiers: int = KeyboardModifier.NONE, device_id: int = 1*) → None

Sends a keyboard device event to the server.

Arguments:

key_code: keyboard key code (ASCII character, numerical value, enum *Server.KeyboardServerKeys*)

modifiers: keyboard modifier key identifier (enum *Server.KeyboardModifier*)

device_id: keyboard device identifier

Command identifier: PUSH_KEYBOARD

cmd_push_cursor(*x: int = -1, y: int = -1, button: int = PointerButton.NONE, state: int = PointerState.MOVE, modifiers: int = PointerModifier.NONE, device_id: int = 1, du_id: int = 1, window_idx: int = 0*) → None

Sends a cursor device event to the server.

Arguments:

x: X origin position of the cursor in pixels (-1 means keep same position)

y: Y origin position of the cursor in pixels (-1 means keep same position)

button: cursor button identifier (enum *Server.PointerButton*)

state: pointer state (enum *Server.PointerState*)

modifiers: modifier key identifier (enum *Server.PointerModifier*)

device_id: cursor device identifier

du_id: DU identifier

window_idx: Window index

Command identifier: PUSH_CURSOR

cmd_push_touch(*x: int = 0, y: int = 0, state: int = TouchState.MOVE, timestamp: int = 0, touch_point_id: int = 1, device_id: int = 1, window_idx: int = 0, for_gesture: int = 0*) → None

Sends a touch device event to the server.

Arguments:

x: X origin position of the touch point in pixels

y: Y origin position of the touch point in pixels

state: touch point state (enum Server.TouchState)

timestamp: timestamp in micro-seconds

touch_point_id: touch point identifier

device_id: touch device identifier

window_idx: window index

for_gesture: When set to a value different from 0, the command is used to drive the gesture recognizer. 1 means that the touch is sent to the gesture recognizer touch queue, and 2 is used to force the computation of the gesture recognizer (the touch data is not used in this case).

Command identifier: PUSH_TOUCH

cmd_set_cursor_active_state(*cursor: int = 0, active: bool = True*) → None

Sets cursor active state.

Arguments:

cursor: cursor number (from 1 to the number of cursor pointers declared in the server conf). 0 means no change (used to get the list of active cursors)

active: cursor active state

Command identifier: SET_CURSOR_ACTIVE_STATE

cmd_set_interface_value(*app_id: int, layer_id: int, widget_id: int, interface_name: str, method_name: str, value: int*) → None

For a server/widget interface, the function is used to force the value of a given method (interface input).

The supported interfaces/methods are:

- ForceEvent/ForceEvent (value=message ID)
- Keyboard/PredefKeys (value=ulong range)
- PopUpMenu/CloseRequest (value=0/1)
- Selection/Select (value=0/1)
- UAState/NoService (value=0/1)

Arguments:

app_id: user application (UA) identifier

layer_id: layer identifier

widget_id: widget identifier

interface_name: interface name

method_name: method name for the given interface

value: value to set (integer value)

Command identifier: SET_INTERFACE_VALUE

cmd_step(*steps: int = 1, until_has_blocks: bool = False, interactive: bool = False*) → None

Controls the cycle execution of the server.

Arguments:

steps:

- When set to a positive value, it will start the cycle execution for a maximum number of cycle steps.
- When set to -1, it will start the cycle execution for an unlimited number of cycle steps.
- When set to 0, it will stop the cycle execution.

until_has_blocks: when set to True, runs until the server produces an A661 notification (excluding the server control proxy notifications).

interactive:

- When set to False, the server runs with all physical input devices disabled.
- When set to True, the physical input devices are enabled.

Command identifier: STEP

cmd_server_exit() → None

Notifies the server that it is shutting down.

Command identifier: SERVER_EXIT

cmd_coverage_reset() → None

Resets coverage in the instrumented server.

Command identifier: COVERAGE_RESET

cmd_coverage_dump(*pathname: str*) → None

Dumps coverage in the instrumented server.

Arguments:

pathname: file path without extension, used as basename for dump of raw coverage files (.info and .sdyinfo).

Command identifier: COVERAGE_DUMP

Common Classes

Server Class

class **Server**(*exe: str = 'example/A661Server.exe', port: int = 0, udp: bool = False, name: str = '', df_list: Iterable[str] = None, conf: Optional[str] = None, datagroup: bool = False*)

A661 Server class.

Arguments

exe: path on the server binary

port: base socket port used to communicate between server and UAs (by default 1230)

udp: UDP/IP protocol when True, TCP/IP protocol when False (default is False)

name: Name displayed in header of all DU windows (by default "A661Server")

exe: = name of the server executable file

df_list: List of Dfs to be loaded by the server

conf: Configuration file to be loaded with the server

datagroup: Use datagroup protocol when set to True (default is False)

class **PointerState** *(value)*

Cursor pointer device current state.

class **PointerButton** *(value)*

Cursor pointer device selected button.

class **PointerModifier** *(value)*

Cursor pointer device selected modifier.

class **KeyboardModifier** *(value)*

Keyboard device selected modifier.

class **KeyboardServerKeys** *(value)*

Keyboard device keycodes for predefined keys.

class **TouchState** *(value)*

Touch point device current state.

class **GestureType** *(value)*

Gesture device current gesture type.

class **GestureState** *(value)*

Gesture device current gesture state.

run(*console: bool = True, debug: str = '', logfile: str = 'server_trace.log', pause: bool = False, extra: str = ''*) → Server

Launches server.

Arguments

console: True to use a separate MS-DOS console to launch the server.

debug: not used

logfile: the name of output file for server log traces

pause: (only when console is set to True) wait confirmation to close the console when the server ends.

extra: extra parameters to be passed to the server.

close(*delay: int = 5*) → Server

Ends server execution.

Arguments

delay: timeout in seconds before forcing the server to close (-1 means close is not forced)

UA Class

class **UA**(*ua_id: int*, *udp: bool = False*)

User Application class.

Arguments

ua_id: UA identifier

udp: True to set-up an UDP connection with server (otherwise it is TCP). Used by connect method.

Typical usage:

- init+socket connection:

```
ua = UA(<ua_id>)
ua.connect()
```

- register Df object:

```
ua.register_df(df)
```

- send message to server:

```
ua.send_block(<list of Widget, WidgetExtension and Layer objects>)
```

- get notificiations sent by server:

```
notifs = ua.get_notifications()
```

connect(*server_ip: str = '127.0.0.1'*, *base_port: int = 1230*) → UA

Sets up the connection with the server.

Arguments

server_ip: server hostname (default is “127.0.0.1”)

base_port: socket base port (default is 1230). The base port is used to calculate the UA port: UA port = base port + UA Id.

Returns

The current UA object.

register_df(*df_object: Df*) → UA

Associates the symbols, pictures and layers (with corresponding widgets) of a Df to this UA.

Arguments

df_object: Df object

Returns

The current UA object.

get_widget(*layer_id: int, widget_id: int*) → Optional[Widget]

Returns registered widget object according to its layer and widget id.

Arguments

layer_id: Id of layer containing the requested widget.
widget_id: Id of requested widget.

Returns

The requested widget object, or None if not found.

get_layer(*layer_id: int*) → Optional[Layer]

Returns registered layer object according to its layer id.

Arguments

layer_id: Id of the requested layer.

Returns

The requested layer object, or None if not found.

send_block(*data: List[Layer | Widget | ExtensionWidget] | Layer | Widget | ExtensionWidget | Msg | bytes | str, layer_id: Layer | int | None = None, context_nb: int = 0*) → None

Sends an A661 block to the server.

Arguments

data: the data sent to the server which can correspond to:

- a Widget or ExtensionWidget object to build a set parameter command
- a Layer object to build a layer request
- a list of Widget, ExtensionWidget and Layer objects
- an Msg object or a bytes type to send a raw block to the server
- a string corresponding to raw value in hexadecimal (*e.g.* “CA02FA0010” or “CA.02.FA.00.10”)

layer_id: layer identifier (only used if no Layer object in data)

context_nb: context number

Returns

None

When *data* is a Msg object, a bytes or an hexa string, it corresponds to the full raw A661 block, starting with A661_BEGIN_BLOCK. or a string representing the data in hexadecimal (with or without ‘.’ separator between each byte). *layer* and *context_nb* parameters are ignored in this case (as they are part of the A661 block).

Otherwise, the A661 block is built as follows:

- For a set parameter command (A661_CMD_SET_PARAMETER):

  <widget object>(<widget_id>).<message name>(<message value>)

  The message name is the name of the parameter ident in lower case without A661_ prefix.

  For example, to build message A661_STYLE_SET to #110 for ToggleButton widget #3:

```
ToggleButton(3).style_set(110)
```

- For a layer request:

  Layer(<layer_id>).<message name>(<message value>...)

  The message name is the name of the layer request in lower case without A661_ prefix.

  For example, to build message A661_REQ_LAYER_ACTIVE for layer #1:

  ```
  Layer(1).req_layer_active()
  ```

An A661 block is sent to a given layer. It is not possible to provide Layer objects with different ids in *data*. The layer id is provided by the *layer_id* parameter or the id of any Layer object provided in *data*. If there is no Layer object in data and *layer_id* is not set, then the A661 block is sent to layer #1.

send_datagroup*(data: List[Tuple[List[Layer | Widget | ExtensionWidget] | Layer | Widget | ExtensionWidget | Msg | bytes | str, int, int]])* → None

Sends a datagroup of A661 blocks to the server.

Arguments

data: a list of tuples. Each tuple is composed of an A661 block, its associated layer id and the context number of the layer.

- the A661 block data follows the same rules as for the function *send_block*.
- the layer id and the context number are used according to A661 block format.

Returns

None

get_notifications(*number_of_blocks: int = 0, non_blocking: bool = True*) → List[Dict[str, Any]]

Gets pending CDS notifications sent by the server.

Arguments

number_of_blocks: Indicate how many notification blocks or datagroups are considered. 0 means all pending blocks or datagroups.

non_blocking: When number_of_blocks is not 0, set it to False to wait until the number of requested blocks or datagroups is reached.

Returns

A list of A661 CDS notifications, or an empty list if no notification available.

Each notification is a dictionary, which contains at least the following information:

```
{
   "app_id": *application id*,
   "layer_id": *layer id*,
   "context": *context number of the layer*,
   "kind": *notification kind*,
   "notif": *notification content according to kind*
}
```

notification kind is one of the following:

- Widget Event (A661_NOTIFY_WIDGET_EVENT):
  The following keys are also present in the dictionary:
  - "widget_id": widget id
  - "origin": notification origin (always equals to 1 in current implementation)

  "*notif*" is an event notification object <event>

  When a Df object is provided, <event> object will contain the links to the corresponding layer and widget objects.
- Layer (A661_NOTIFY_LAYER_EVENT):
  "*notif*" is a dictionary with "*event*" key giving the event id and "*value*" key giving the event value.
- Exception (A661_NOTIFY_EXCEPTION):
  "*notif*" is a dictionary with "*event*" key giving the event id and "*value*" key giving a tuple of 3 elements:
  - the reason of the exception (tuple with error code and error name)
  - the widget id at the origin of the exception (when applicable, otherwise 0)
  - the parameter involved in the exception (when applicable tuple with parameter id and name, otherwise (0, NONE))

Df Class

class **Df**(*ua_id: int, layers: Iterable[Layer] = None, charset_encoding: Optional[CharsetEncoding] = None, picture_blocks: Iterable[PictureBlock] = None, symbol_blocks: Iterable[SymbolBlock] = None*)

Definition File class.

Arguments

ua_id: UA Identifier

layers: List of layers (Layer object), if any.

charset_encoding: Charset encoding (CharsetEncoding object). Default is ASCII.

picture_blocks: List of Picture blocks (PictureBlock object), if any.

symbol_blocks: List of Symbol blocks (SymbolBlock object), if any.

write(*file_name: str, oem_free: bytes = b''*) → None

Method to serialize the Df object as a Df binary file.

Arguments

file_name: File name of the resulting binary Df.

oem_free: OEM data to be added to the Df.

Returns

None

write_hex(*file_name: str, oem_free: bytes = b''*) → None

Method to serialize the Df object as a Df hexadecimal file.

Arguments

file_name: File name of the resulting hexa Df.

oem_free: OEM data to be added to the Df.

Returns

None

CharsetEncoding Class

class **CharsetEncoding**(*encod: str*)

Charset Encoding class

Arguments

encod: A661 constant representing the encoding, either “A661_UTF8” or “A661_GBK”.

PictureBlock Class

class **PictureBlock**(*pblock_id: int, pixel_format: str | int, color_format: str | int, color_tab: List[Any] = [], children: List[PictureInc] = []*)

Picture Block class (corresponds to A661_Picture_Block_Structure_DT structure)

Arguments

pblock_id: Picture block index (value used by UA class to register the picture blocks).

pixel_format: PixelFormat parameter (integer value or string corresponding to expected A661 constant).

possible values are: ‘A661_PIX_FMT_RGBA_8’, ‘A661_PIX_FMT_LUMINANCE_ALPHA_8’ or ‘A661_PIX_FMT_COLOR_INDEXED_8’

color_format: ColorTableFormat parameter (integer value or string corresponding to expected A661 constant).

possible values are: ‘A661_PIX_FMT_RGBA_8’ or ‘A661_PIX_FMT_LUMINANCE_ALPHA_8’

color_tab: Color table if any (list of values depending on color_format).

children: List of Pictures (PictureInc object). Can be later added using the add_pics method.

add_pic(*child: PictureInc*) → PictureBlock

Adds a Picture (PictureInc object) to the Picture Block.

add_pics(*pics: List[PictureInc]*) → PictureBlock

Adds a List of Pictures (PictureInc object) to the Picture Block.

PictureInc Class

class **PictureInc**(*picture_reference: int, width: int, height: int, pixel_values: List[Any]*)

Picture class (corresponds to A661_Picture_Structure_DT structure)

Arguments

picture_reference: PictureReference parameter: picture identifier

width: NumberOfPixelsWidth parameter: picture width in pixels

height: NumberOfPixelsHeight parameter: picture height in pixels

pixel_values: Picture content: list of pixel values depending on PixelFormat

SymbolBlock Class

class **SymbolBlock**(*sblock_id: int, children: List[Symbols] = None*)

Symbol Block class (corresponds to A661_Symbol_Block_Structure_DT structure)

Arguments

sblock_id: Symbol block index (value used by UA class to register the symbol blocks).

children: List of Symbols (Symbols object). Can be later added using the add_symbs method.

add_sym(*child: Symbols*) → SymbolBlock

Adds a Symbol (Symbols object) to the Symbol Block.

add_syms(*children: List[Symbols]*) → SymbolBlock

Adds a list of Symbols (Symbols object) to the Symbol Block.

Symbols Class

class **Symbols**(*symbol_id: int, children: List[SymbolCmd] = None*)

Symbol definition class (corresponds to A661_Create_Symbol_Structure structure)

Arguments

symbol_id: SymbolId parameter: symbol identifier

children: List of symbol commands (SymbolCmd object). Can be later added using the add_symbol_incs method.

add_symbol_inc(*child: SymbolCmd*) → Symbols

Adds a symbol command (SymbolCmd object) to the Symbol.

add_symbol_incs(*symbols: List[SymbolCmd]*) → Symbols¶

Adds a list of symbol commands (SymbolCmd object) to the Symbol.

SymbolCmd Class

class **SymbolCmd**(*children: List[SymbolCmd] = None*)

Symbol command definition class (generic class, see list of available symbol commands in dedicated paragraph).

Arguments

children: List of children symbol commands (SymbolCmd object). Can be later added using the add_symcmds method.

add_symcmd(*child: SymbolCmd*) → SymbolCmd

Adds a child symbol command (SymbolCmd object) to the current symbol command.

add_symcmds(*children: List[SymbolCmd]*) → SymbolCmd¶

Adds a list of children symbol commands (SymbolCmd object) to the current symbol command.

Layer Class

class **Layer**(*layer_id: int, context_nb: int = 0, children: List[Widget] | None = None*)

Layer class

Arguments

layer_id: Layer identifier.

context_nb: Context number.

children: List of widgets (Widget object). Can be later added using the add_widgets method.

add_widget(*child: Widget*) → Layer

Adds a widget (Widget object) to the Layer.

add_widgets(*children: List[Widget]*) → Layer

Adds a list of widgets (Widget object) to the Layer.

req_layer_visible() → Layer

Method to build the UA request A661_REQ_LAYER_VISIBLE.

req_layer_active() → Layer

Method to build the UA request A661_REQ_LAYER_ACTIVE.

req_layer_inactive() → Layer

Method to build the UA request A661_REQ_LAYER_INACTIVE.

req_focus_on_widget(*widget_id: int*) → Layer

Method to build the UA request A661_REQ_FOCUS_ON_WIDGET.

req_cursor_on_widget(*widget_id: int, oem_data: int = 0*) → Layer

Method to build the UA request A661_REQ_CURSOR_ON_WIDGET.

Widget Class

class **Widget**(*widget_id: int | Counter, children: List[Widget] | None = None, extensions: List[ExtensionWidget] | None = None*)

Widget class (generic class, see list of available widgets in dedicated paragraph).

Arguments

widget_id: Widget identifier.

children: List of children widgets (Widget object). Can be later added using the add_widgets method.

extensions: List of widget extensions associated to the widget (ExtensionWidget object). Can be later added using the add_extensions method.

add_widget(*child: Widget*) → Widget

Adds a child widget (Widget object) to the current widget.

add_widgets(*children: List[Widget]*) → Widget

Adds a list of children widgets (Widget object) to the current widget.

add_extension(*exten: ExtensionWidget*) → Widget

Adds a widget extension (ExtensionWidget object) to the current widget.

add_extensions(*extensions: List[ExtensionWidget]*) → Widget

Adds a list of widget extensions (ExtensionWidget object) to the current widget.

ExtensionWidget Class

class **ExtensionWidget**(*parent_id: int = 0*)

Widget extension class (generic class, see list of available widget extensions in dedicated paragraph).

Arguments

parent_id: Associated widget identifier.

Utility Functions

create_df_from_sgfx

create_df_from_sgfx(*sgfx_path: str | List[str], a661_path: str = '', picture_path: str | List[str] | None = None, symbol_path: str | List[str] | None = None, ua_id: int = 1*) → Df

Creates a DF object from UAPC model file(s).

Arguments

sgfx_path: the path to the UAPC DF model file (.sgfx). A list of models can be provided.

a661_path: the path on the main A661 description file. Default is `<SCADEDISPLAY_HOME>/SCADE A661/a661_description/a661.xml`

picture_path: Optional path on the UAPC A661 picture table file (.pdt). A list of picture tables can be provided.

symbol_path: Optional path on the UAPC A661 symbol table file (.sdt). A list of symbol tables can be provided.

ua_id: The UA identifier for the created DF object. Default is 1.

Returns

The Df object.

generate_binary_hex

generate_binary_hex(*df_object: Df, output_file: str = 'UA', output_path: str = '.', oem_data: bytes = b''*) → None

Creates a binary DF file and its hexadecimal representation from a Df object.

Arguments

df_object: The Df object

output_file: The file name (without suffix) for the generated DF files. Default is 'UA'.

output_path: The directory path for the generated DF files. Default is '.'.

oem_data: When present, add the corresponding bytes in the OEM_Free_Data section of the binary DF.

Returns

None

gen_sgfx_from_binary

gen_sgfx_from_binary(*df_file: str, output_path: str = '', a661_path: str = '', dfimport_path: str = ''*) → None

Creates an UAPC model file (.sgfx) from a binary DF file (using dfimport tool).

Arguments

df_file: The path on the DF binary file

output_path: The directory path for the generated sgfx file. Default is same directory as the DF binary file.

a661_path: The path on the main A661 description file. Default is `<SCADEDISPLAY_HOME>/SCADE A661/a661_description/a661.xml`

dfimport_path: The path on the dfimport tool. Default is `<SCADEDISPLAY_HOME>/SCADE A661/bin/dfimport.exe`

Returns

None

Index

Index

Index