Industrial Controls
Engineering Department

UNICOS Application Builder

User Template API for Developers

v1.9.0

07/12/2016

Table Of Content

Interface IS7SymbolTemplate

< Methods >

public interface **IS7SymbolTemplate**

Interface to provide methods to access the Step 7 symbols. The interface is implemented by the Siemens Step 7 plug-ins (S7CodeGenerator and S7LogicGenerator)

Author:

Ivan Prieto Barreiro

Methods

s7db_id

```
public java.lang.String s7db_id(java.lang.String name)
                        throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(AIRinstance1Name) returns DB_AIR_All.AIR_SET.

FeedbackOn = instance.getAttributeData ("FEDeviceEnvironmentInputs:Feedback On")

if FeedbackOn != "":

s7db_id_result=self.thePlugin.s7db_id(FeedbackOn)

self.thePlugin.writeInstanceInfo('''

$Name$.HFOn:="'+s7db_id_result+FeedbackOn+'".PosSt;"')

Parameters:

name - Is the name of the instance that we want to process

Returns:

The symbol name.

Throws:

java.lang.Exception -

s7db_id

```
public java.lang.String s7db_id(java.lang.String name,
                                 boolean isDBSimpleRequested)
                          throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(AIRinstance1Name,false) returns DB_AIR_All.AIR_SET.

NOTE:
- If the flag isDBSimpleRequested is TRUE, and
- If the specified instance is an AnalogInputReal or a DigitalInput device

Then, the returned String will be
"DB_<RepresentationName>_All_S.<RepresentationName>_SET."

```
s7db_id_result=self.thePlugin.s7db_id("DigitalInput_1")
```

Parameters:

name - Is the name of the instance that we want to process
isDBSimpleRequested - TRUE if the DB Simple is requested.

Returns:

The symbol name.

Throws:

java.lang.Exception -

---

s7db_id

```
public java.lang.String s7db_id(java.lang.String name,
                                 java.lang.String deviceTypes)
                          throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(AIRinstance1Name,"AnalogInputReal") returns DB_AIR_All.AIR_SET.

```
FeedbackOn = instance.getAttributeData("FEDeviceEnvironmentInputs:Feedback On")
if FeedbackOn != "":
    s7db_id_result=self.thePlugin.s7db_id(FeedbackOn, "DigitalInput")
    self.thePlugin.writeInstanceInfo('''
    $Name$.HFOn:="'+s7db_id_result+FeedbackOn+'".PosSt;"')
```

Parameters:

name - Is the name of the instance that we want to process
deviceTypes - Device type list (comma separated)

Returns:

The symbol name.

Throws:

java.lang.Exception -

s7db_id

```
public java.lang.String s7db_id(java.lang.String name,
                                java.lang.String deviceTypes,
                                boolean isDBSimpleRequested)
                         throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(AIRinstance1Name,"AnalogInputReal") returns DB_AIR_All.AIR_SET.

NOTE: If the flag isDBSimpleRequested is TRUE and:
- If the flag isDBSimpleRequested is TRUE, and
- If the specified instance is an AnalogInputReal or a DigitalInput device

Then, the returned String will be
"DB_<RepresentationName>_All_S.<RepresentationName>_SET."

FeedbackOn = instance.getAttributeData ("FEDeviceEnvironmentInputs:Feedback On")

if FeedbackOn != "":

s7db_id_result=self.thePlugin.s7db_id(FeedbackOn, "DigitalInput")

self.thePlugin.writeInstanceInfo("'

$Name$.HFOn:="'+s7db_id_result+FeedbackOn+"'.PosSt;"')

Parameters:

name - Is the name of the instance that we want to process
deviceTypes - Device type list (comma separated)
isDBSimpleRequested - TRUE if the DB Simple is requested. * @return The symbol name.

Returns:

The symbol name.

Throws:

java.lang.Exception -

s7db_id

```
public java.lang.String
s7db_id(research.ch.cern.unicos.utilities.IDeviceInstance instance,
                                boolean isDBSimpleRequested)
                         throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(AIRinstance1,false) returns DB_AIR_All.AIR_SET.

NOTE:
- If the flag isDBSimpleRequested is TRUE, and
- If the specified instance is an AnalogInputReal or a DigitalInput device

Then, the returned String will be
"DB_<RepresentationName>_All_S.<RepresentationName>_SET."

FeedbackOn = instance.getAttributeData ("FEDeviceEnvironmentInputs:Feedback On")

if FeedbackOn != "":

s7db_id_result=self.thePlugin.s7db_id(FeedbackOn, true)

self.thePlugin.writeInstanceInfo('''

$Name$.HFOn:='"+s7db_id_result+FeedbackOn+'".PosSt;'")

Parameters:

instance - Is the instance which symbol is requested.
isDBSimpleRequested - TRUE if the DB Simple is requested.

Returns:

The symbol name.

Throws:

java.lang.Exception -

Interface ISCADAPlugin

< Methods >

---

public interface **ISCADAPlugin**

Interface to be implemented by all the CPC SCADA plug-ins.

Author:

Ivan Prieto Barreiro

Methods

computeAddress

`public java.lang.String` **`computeAddress`**`(java.lang.String deviceAlias)`

Get the address of a device element (e.g. computeAddress("AnalogInput1_MPosR"))

Parameters:

deviceAlias - The device aliase.

Returns:

The requested address if it exists, otherwise the empty String.

---

getPLCMemoryMapper

`public research.ch.cern.unicos.utilities.IPLCMemoryMapper` **`getPLCMemoryMapper`**`()`

Get the PLC memory mapper instance.

Returns:

The PLC memory mapper instance.

---

getResourceAddress

`public java.lang.String` **`getResourceAddress`**`(java.lang.String resource)`

For Siemens only: get the DB number for a given resource name, e.g.
DB_WINCCOA_Address = self.thePlugin.getResourceAddress("DB_WINCCOA")
thePlcDsIpAddress = "DB"+str(DB_WINCCOA_Address)+".DBD0"

Parameters:

resource - The resource which address is required.

Returns:

The requested DB number for the resource if it exists, otherwise null.

---

writeInstanceInfo

`public void` **`writeInstanceInfo`**`(java.lang.String theInstanceData)`

Used to write an instance declaration in the WinCC O.A. Data file

Parameters:

theInstanceData -

Interface IUnityLogicTemplateFunctions

< Methods >

public interface **IUnityLogicTemplateFunctions**

Interface containing the methods that can be used in the user logic templates for the UnityLogicGenerator.

Author:

Ivan Prieto Barreiro

Methods

writeProgram

```
public void writeProgram(java.lang.String theProgram)
```

Used to write a part of the user logic application.

Parameters:

theProgram - The program to be written. **Example:**

```
# Write a part of the program self.thePlugin.writeProgram('''(*
Adding a comment to the generated application *)''')
```

writeVariable

```
public void writeVariable(java.lang.String theVariable)
```

Used to write a variable in the dedicated buffer.

Parameters:

theVariable - The variable to be written.

Example:

```
# Write Variables self.thePlugin.writeVariable(''' ''')
```

Class S7Functions

```
java.lang.Object
    |
    +--research.ch.cern.unicos.cpc.utilities.siemens.S7Functions
```

< Constructors > < Methods >

public class **S7Functions**
extends java.lang.Object

Class containing functions for Siemens Step 7

Author:

Ivan Prieto Barreiro

Constructors

S7Functions

```
public  S7Functions()
```

Methods

getTargetDeviceInformationParam

```
public static java.lang.String
getTargetDeviceInformationParam(java.lang.String parameterName,
java.lang.String typeName)
```

Get the value of an attribute defined in the TargetDeviceInformation family from the Device type definition.

Parameters:

parameterName - Name of the parameter in the TargetDeviceInformation family.
typeName - Device type name.

---

initialize

```
public static void initialize()
```

Initialize the value of the internal member 'isLargeApplication'. The value is taken from the UnicosApplication.xml file, parameter: 'SiemensPLC:SiemensSpecificParameters:GeneralConfiguration:LargeApplication'. The method must be called once for each application generation before calling any of the s7dbid(...) class methods.

---

isLargeApplication

```
public static boolean isLargeApplication()
```

Get the value of the class member isLargeApplication @see {@link #initialize()}

Returns:

The value of the class member 'isLargeApplication' obtained in the method {@link #initialize()}.

---

s7db_id

```
public static java.lang.String s7db_id(java.lang.String name)
                               throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(specFile, AIRinstance1Name) returns DB_AIR_All.AIR_SET.

The function will use the same specs. file used in the application to search for the specified instance.

Calling this method is equivalent to calling: {@link #s7db_id(IInstancesFacade, String, boolean)} with the boolean parameter equal to false.

Example:

```
FeedbackOn = instance.getAttributeData
("FEDeviceEnvironmentInputs:Feedback On")
if FeedbackOn != "":
    s7db_id_result=self.thePlugin.s7db_id(FeedbackOn)
    self.thePlugin.writeInstanceInfo('''
    $Name$.HFOn:='''+s7db_id_result+FeedbackOn+'''.PosSt;''')
```

Parameters:

name - Is the name of the instance that we want to process

Throws:

java.lang.Exception -

---

s7db_id

```
public static java.lang.String s7db_id(java.lang.String name,
                                       boolean isDBSimpleRequested)
                                throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(specFile, AIRinstance1Name, false) returns DB_AIR_All.AIR_SET. The function will use the same specs. file used in the application to search for the specified instance.

NOTE:
- If the flag isDBSimpleRequested is TRUE, and
- If the specified instance is an AnalogInputReal or a DigitalInput device

Then, the returned String will be
"DB_<RepresentationName>_All_S.<RepresentationName>_SET."

Example:

```
FeedbackOn = instance.getAttributeData
("FEDeviceEnvironmentInputs:Feedback On")
if FeedbackOn != "":
    s7db_id_result=self.thePlugin.s7db_id(self.theRawInstances,
FeedbackOn, false)
    self.thePlugin.writeInstanceInfo('''
    $Name$.HFOn:='''+s7db_id_result+FeedbackOn+'''.PosSt;''')
```

Parameters:

name - Is the name of the instance that we want to process
isDBSimpleRequested - TRUE if the DB Simple is requested.

Throws:

java.lang.Exception -

s7db_id

```
public static java.lang.String s7db_id(java.lang.String name,
                                       java.lang.String deviceTypes)
                                throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(AIRinstance1Name, "AnalogInputReal") returns DB_AIR_All.AIR_SET. The function will use the same specs. file used in the application to search for the specified instance. Calling this method is equivalent to: {@link #s7db_id(IInstancesFacade, String, String, boolean)} with the boolean parameter equal to false.

Example:

```
FeedbackOn =
instance.getAttributeData("FEDeviceEnvironmentInputs:Feedback On")
if FeedbackOn != "":
    s7db_id_result=self.thePlugin.s7db_id(FeedbackOn, "DigitalInput")
    self.thePlugin.writeInstanceInfo('''
    $Name$.HFOn:='''+s7db_id_result+FeedbackOn+'''.PosSt;''')
```

Parameters:

name - Is the name of the instance that we want to process
deviceTypes - Device type list (comma separated)

Throws:

java.lang.Exception -

s7db_id

```
public static java.lang.String s7db_id(java.lang.String name,
                                       java.lang.String deviceTypes,
                                       boolean isDBSimpleRequested)
                                throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(AIRinstance1Name,"AnalogInputReal", false) returns DB_AIR_All.AIR_SET.

The function will use the same specs. file used in the application to search for the specified instance.

NOTE:
- If the flag isDBSimpleRequested is TRUE, and
- If the specified instance is an AnalogInputReal or a DigitalInput device

Then, the returned String will be
"DB_<RepresentationName>_All_S.<RepresentationName>_SET."

Example:

```
FeedbackOn =
instance.getAttributeData("FEDeviceEnvironmentInputs:Feedback On")
if FeedbackOn != "":
    s7db_id_result=self.thePlugin.s7db_id(FeedbackOn, "DigitalInput",
false)
    self.thePlugin.writeInstanceInfo('''
    $Name$.HFOn:='''+s7db_id_result+FeedbackOn+'''.PosSt;''')
```

Parameters:

name - Is the name of the instance that we want to process
deviceTypes - Device type list (comma separated)
isDBSimpleRequested - TRUE if the DB Simple is requested.

Throws:

java.lang.Exception -

s7db_id

```
public static java.lang.String
s7db_id(research.ch.cern.unicos.utilities.IDeviceInstance instance,
                                        boolean isDBSimpleRequested)
                                 throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(Instance1Name, false) returns DB_AIR_All.AIR_SET.

NOTE:
- If the flag isDBSimpleRequested is TRUE, and
- If the specified instance is an AnalogInputReal or a DigitalInput device

Then, the returned String will be
"DB_<RepresentationName>_All_S.<RepresentationName>_SET."

Parameters:

instance - The device instance.
isDBSimpleRequested - TRUE if the DB Simple is requested.

Throws:

java.lang.Exception -

---

s7db_id

```
public static java.lang.String
s7db_id(research.ch.cern.unicos.utilities.IInstancesFacade theUnicosProject,
                                        java.lang.String name)
                                 throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(specFile, AIRinstance1Name) returns DB_AIR_All.AIR_SET. Calling this method is equivalent to calling: {@link #s7db_id(IInstancesFacade, String, boolean)} with the boolean parameter equal to false.

Example:

```
FeedbackOn = instance.getAttributeData
("FEDeviceEnvironmentInputs:Feedback On")
if FeedbackOn != "":
    s7db_id_result=self.thePlugin.s7db_id(self.theRawInstances,
FeedbackOn)
    self.thePlugin.writeInstanceInfo('''
    $Name$.HFOn:='''+s7db_id_result+FeedbackOn+'''.PosSt;''')
```

Parameters:

theUnicosProject - Java representation of the UNICOS specifications file.
name - Is the name of the instance that we want to process

Throws:

java.lang.Exception -

s7db_id

```
public static java.lang.String
s7db_id(research.ch.cern.unicos.utilities.IInstancesFacade theUnicosProject,
                                        java.lang.String name,
                                        boolean isDBSimpleRequested)
                                 throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(specFile, AIRinstance1Name, false) returns DB_AIR_All.AIR_SET.

NOTE:
- If the flag isDBSimpleRequested is TRUE, and
- If the specified instance is an AnalogInputReal or a DigitalInput device

Then, the returned String will be "DB_<RepresentationName>_All_S.<RepresentationName>_SET."

Example:

```
FeedbackOn = instance.getAttributeData
("FEDeviceEnvironmentInputs:Feedback On")
if FeedbackOn != "":
    s7db_id_result=self.thePlugin.s7db_id(self.theRawInstances,
FeedbackOn, false)
    self.thePlugin.writeInstanceInfo('''
    $Name$.HFOn:='''+s7db_id_result+FeedbackOn+'''.PosSt;''')
```

Parameters:

theUnicosProject - Java representation of the UNICOS specifications file.
name - Is the name of the instance that we want to process
isDBSimpleRequested - TRUE if the DB Simple is requested.

Throws:

java.lang.Exception -

s7db_id

```
public static java.lang.String
s7db_id(research.ch.cern.unicos.utilities.IInstancesFacade theUnicosProject,
                                        java.lang.String name,
                                        java.lang.String deviceTypes)
                                 throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(specFile, AIRinstance1Name, "AnalogInputReal") returns DB_AIR_All.AIR_SET. Calling this method is equivalent to: {@link #s7db_id(IInstancesFacade, String, String, boolean)} with the boolean parameter equal to false.

Example:

```
FeedbackOn =
instance.getAttributeData("FEDeviceEnvironmentInputs:Feedback On")
if FeedbackOn != "":
    s7db_id_result=self.thePlugin.s7db_id(self.theRawInstances,
FeedbackOn, "DigitalInput")
    self.thePlugin.writeInstanceInfo('''
    $Name$.HFOn:='''+s7db_id_result+FeedbackOn+'''.PosSt;''')
```

Parameters:

theUnicosProject - Java representation of the UNICOS specifications file.
name - Is the name of the instance that we want to process
deviceTypes - Device type list (comma separated)

Throws:

java.lang.Exception -

s7db_id

```
public static java.lang.String
s7db_id(research.ch.cern.unicos.utilities.IInstancesFacade theUnicosProject,
                                        java.lang.String name,
                                        java.lang.String deviceTypes,
                                        boolean isDBSimpleRequested)
                                 throws java.lang.Exception
```

This function provides the symbol name of the DB for a given instance name e.g. s7db_id(specFile, AIRinstance1Name,"AnalogInputReal", false) returns DB_AIR_All.AIR_SET.

NOTE:
- If the flag isDBSimpleRequested is TRUE, and
- If the specified instance is an AnalogInputReal or a DigitalInput device

Then, the returned String will be "DB_<RepresentationName>_All_S.<RepresentationName>_SET."

```
FeedbackOn = instance.getAttributeData
("FEDeviceEnvironmentInputs:Feedback On")
if FeedbackOn != "":
    s7db_id_result=self.thePlugin.s7db_id(self.theRawInstances,
FeedbackOn, "DigitalInput", false)
    self.thePlugin.writeInstanceInfo('''
    $Name$.HFOn:='''+s7db_id_result+FeedbackOn+'''.PosSt;''')
```

Parameters:

theUnicosProject - Java representation of the UNICOS specifications file.
name - Is the name of the instance that we want to process
deviceTypes - Device type list (comma separated)
isDBSimpleRequested - TRUE if the DB Simple is requested.

Throws:

java.lang.Exception -

Interface IGenerationPluginTemplate

< Methods >

public interface **IGenerationPluginTemplate**

This interface provides the common methods used in the Jython templates
to interact with the generation plug-ins.

Author:

Ivan Prieto Barreiro

Methods

formatNumberPLC

```
public java.lang.String formatNumberPLC(java.lang.String excelParameter)
```

Method used to get the correct number format for the PLC. If the parameter is a float number applies the float format, otherwise returns the same value provided.

Parameters:

excelParameter - Value coming from the specs

---

getApplicationName

```
public java.lang.String getApplicationName()
```

Get the application name from the UnicosApplication.

Returns:

The application name.

---

getLinkedExpertName

```
public java.lang.String getLinkedExpertName(java.lang.String theName)
                        throws null
```

Function. it's used to get the Expert Name (if it exists).

Parameters:

theName - Name of the device instance to look for.

Returns:

If the instance doesn't exist returns 'theName' given as first argument. If the instance has an expert name, the expert name is returned, otherwise returns the instance name.

Throws:

null - If the specs file is not defined.

---

getPlcManufacturer

```
public java.lang.String getPlcManufacturer()
```

Get the PLC manufacturer from the PLC declarations

Returns:

a String containing the name of the PLC manufacturer of the application or null if the PLCManufacturer is not defined.

---

getPlcName

```
public java.lang.String getPlcName()
                        throws null
```

Get the PLC name as defined in the UnicosApplication.xml

Returns:

The PLC name if it's defined, otherwise null.

Throws:

null -

---

getUnicosProject

```
public ISpecFileTemplate getUnicosProject()
                                                    throws null
```

Provide access to the root of the input XML data (specs file instance)

Returns:

The instance of the specifications file.

Throws:

null - a GenerationException if the specs file instance is null.

---

isString

```
public boolean isString(java.lang.String parameter)
```

Method used from the templates to know if the specified parameter is a String or a Float

Parameters:

parameter - The parameter to be checked

Returns:

False if the parameter is a float value, otherwise True.

---

writeFile

```
public void writeFile(java.lang.String fileName,
                      java.lang.String content)
```

Write a file in the plug-in output folder.

Parameters:

fileName - The output file name.
content - The output file content.

---

writeXmlFile

```
public void writeXmlFile(java.lang.String fileName,
                         java.lang.String content)
```

Write an XML file in the plug-in output folder. The content of the XML file will be validated to verify the well-formedness.

Parameters:

fileName - The output file name.
content - The output file content.

Interface ILogWriterTemplate

< Methods >

---

public interface **ILogWriterTemplate**

Interface containing the methods that can be used from the Jython templates to write in the user report window.

Author:

Ivan Prieto Barreiro

Methods

writeConfigInUABLog

```
public void writeConfigInUABLog(java.lang.String message)
```

Write a CONFIG message in UAB Log window.

Parameters:

message - The message to be written in the log window.

Example:

```
thePlugin.writeConfigInUABLog("This is a config message")
```

---

writeDebugInUABLog

```
public void writeDebugInUABLog(java.lang.String message)
```

Write a DEBUG message in UAB Log window.

Parameters:

message - The message to be written in the log window.

Example:

```
thePlugin.writeDebugInUABLog("This is a debug message")
```

---

writeErrorInUABLog

```
public void writeErrorInUABLog(java.lang.String message)
```

Write a SEVERE message in UAB Log window. If one or more error messages are written during the generation the exit status of the generation will be FAILURE.

Parameters:

message - The message to be written in the log window.

Example:

```
thePlugin.writeErrorInUABLog("This is an error message")
```

---

writeErrorWithStackTrace

```
public void writeErrorWithStackTrace(java.lang.String message,
                                     java.lang.Throwable e)
```

Used to write an ERROR message in UAB Log window and the stack trace that caused the error.

Parameters:

message - is the message to be written
e - Exception that caused the error

---

writeFineInUABLog

```
public void writeFineInUABLog(java.lang.String message)
```

Write a FINE message in UAB Log window.

Parameters:

message - The message to be written in the log window.

Example:

```
thePlugin.writeFineInUABLog("This is an fine message")
```

writeInUABLog

```
public void writeInUABLog(java.util.logging.Level theLevel,
                          java.lang.String message,
research.ch.cern.unicos.userreport.UserReportGenerator.type theType)
```

Used to write a message in UAB Log window

Parameters:

theLevel -
message - is the message to be written
theType -

writeInfoInUABLog

```
public void writeInfoInUABLog(java.lang.String message)
```

Used to write an INFO message in UAB Log window.

Parameters:

message - The message to be written in the log window.

Example:

```
thePlugin.writeInfoInUABLog("This is an info message")
```

writeWarningInUABLog

```
public void writeWarningInUABLog(java.lang.String message)
```

Used to write a WARNING message in UAB Log window.

Parameters:

message - The message to be written in the log window.

Example:

```
thePlugin.writeWarningInUABLog("This is a warning message")
```

Class AbsolutePathBuilder

```
java.lang.Object
    |
    +--research.ch.cern.unicos.utilities.AbsolutePathBuilder
```

< Constructors > < Methods >

public class **AbsolutePathBuilder**
extends java.lang.Object

Class to build absolute paths from the UnicosApplication.xml parameters.

If the parameter contains to a relative path, the class will build the absolute path using the path of the UnicosApplication.xml file.

Author:

Ivan Prieto Barreiro

Constructors

AbsolutePathBuilder

`public` **`AbsolutePathBuilder`**`()`

Methods

getAbsolutePath

`public static java.lang.String` **`getAbsolutePath`**`(java.lang.String path)`

Build an absolute path from the relative path provided as parameter.

Parameters:

path - Relative path from the application location.

Returns:

---

getApplicationPathParameter

`public static java.lang.String` **`getApplicationPathParameter`**`(java.lang.String paramLocation)`

Method used to get the absolute path of a UnicosApplication.xml application PathParameter. If the PathParameter refers to a relative path, the method will build the full path taking the UnicosApplicationConfig path.

Parameters:

paramLocation - Parameter location in the plug-in parameters of the UnicosApplicationConfig.

Returns:

The absolute path of the PathParameter.

---

getTechnicalPathParameter

```
public static java.lang.String getTechnicalPathParameter(java.lang.String
paramLocation)
```

Method used to get the absolute path of a UnicosApplication.xml technical PathParameter. If the PathParameter refers to a relative path, the method will build the full path taking the UnicosApplicationConfig path.

Parameters:

paramLocation - Parameter location in the plug-in parameters of the UnicosApplicationConfig.

Returns:

The absolute path of the PathParameter.

---

setCoreManager

```
public void setCoreManager(research.ch.cern.unicos.core.CoreManager
coreManager)
```

Class ConvertToString

```
java.lang.Object
    |
    +--research.ch.cern.unicos.utilities.ConvertToString
```

< Methods >

---

public class **ConvertToString**
extends java.lang.Object

Utility class to get a String from different sources

Author:

Ivan Prieto Barreiro

Methods

getString

```
public static java.lang.String getString(java.io.File file)
                                 throws java.io.IOException
```

Get the content of a File as a String.

Parameters:

file - The file where to read the string.

Returns:

Throws:

java.io.IOException -

---

getString

```
public static java.lang.String getString(java.io.InputStream is)
                                 throws java.io.IOException
```

Get the content of an InputStream as a String.

Parameters:

is - The InputStream where to get the String.

Returns:

Throws:

java.io.IOException -

---

getString

```
public static java.lang.String getString(java.lang.String filePath)
                                 throws java.io.IOException
```

Get the content of a file as a String.

Parameters:

filePath - The absolute path of the file.

Returns:

Throws:

java.io.IOException -

Interface IDeviceInstanceTemplate

< Methods >

public interface **IDeviceInstanceTemplate**

Interface used to handle the device type instances available in the specifications file.

The methods described in this interface can be used by the Jython template developers to complete the logic templates.

Note: The examples provided use the Jython syntax.

Author:

Ivan Prieto Barreiro

Methods

doesSpecificationAttributeExist

`public boolean` **`doesSpecificationAttributeExist`**`(java.lang.String attribute)`

Check if an specification attribute exists.

Parameters:

attribute - The attribute to check. E.g. "DeviceIdentification:Name"

Returns:

TRUE if and only if the specification attribute exists

getAttributeData

`public java.lang.String **getAttributeData**(java.lang.String theAttributeIdentifier)`

Returns the data associated with the instance attribute.

Parameters:

theAttributeIdentifier - The instance attribute (e.g. "DeviceIdentification:Name")

Returns:

The data associated with the attribute as defined in the specifications file.

Example: Display the name of all the DigitalInput instances.

```
# Get the DigitalInput device type
diDeviceType = theRawInstances.getDeviceType("DigitalInput")

# Get a vector with all the DigitalInput instances
diInstances = diDeviceType.getAllDeviceTypeInstances()

# Display the name of all the DigitalInput instances in the UAB log
file (as a debug message)
for instance in diInstances:
    thePlugin.writeDebugInUABLog(instance.getAttributeData("DeviceIdenti
```

---

getDeviceType

`public research.ch.cern.unicos.utilities.IDeviceType **getDeviceType**()`

Method used to get the device type of the current instance.

Returns:

The device type of the current instance.

---

getDeviceTypeName

`public java.lang.String **getDeviceTypeName**()`

Method used to get the device type name of the current instance.

Returns:

The device type name of the current instance.

---

getInstanceNumber

`public int` **`getInstanceNumber`**`()`

Get the number of the current instance as defined in the specifications file.

Returns:

The instance number as defined in the specifications file.

Interface IDeviceTypeTemplate

< Methods >

---

public interface **IDeviceTypeTemplate**

Interface used to handle the device types available in the specifications file.

The methods described in this interface can be used by the Jython template developers to complete the logic templates.

Note: The examples provided use the Jython syntax.

Author:

Ivan Prieto Barreiro

Methods

doesSpecificationAttributeExist

`public boolean` **`doesSpecificationAttributeExist`**`(java.lang.String attribute)`

Check if an specification attribute exists.

Parameters:

attribute - The attribute to check. E.g. "DeviceIdentification:Name"

Returns:

TRUE if and only if the specification attribute exists

---

getAllDeviceTypeInstances

`public java.util.List` **`getAllDeviceTypeInstances`**`()`

Method used to get a vector containing all the instances of the device type.

Returns:

A vector containing all the instances of the device type.

Example: Display the name of all the DigitalInput instances.

```
# Get the DigitalInput device type
diDeviceType = theRawInstances.getDeviceType("DigitalInput")

# Get a vector with all the DigitalInput instances
diInstances = diDeviceType.getAllDeviceTypeInstances()

# Display the name of all the DigitalInput instances in the UAB log
file (as a debug message)
for instance in diInstances:
    thePlugin.writeDebugInUABLog(instance.getAttributeData("DeviceIdenti
```

---

getDescription

`public java.lang.String` **`getDescription`**`()`

Get the device type description as specified in the device type sheet.

Returns:

A String containing the device type description if exists, otherwise an empty string.

---

getDeviceTypeInstance

```
public research.ch.cern.unicos.utilities.IDeviceInstance
getDeviceTypeInstance(int instanceNumber)
```

Method used to get a device type instance by its instance number.

Parameters:

instanceNumber - The instance number.

Returns:

The requested instance object as specified by the user if it exists, otherwise null.

Example: Get the instance number 5 of the DigitalInput device type.

```
# Get the DigitalInput device type
diDeviceType = theRawInstances.getDeviceType("DigitalInput")

# Get the DigitalInput instance number 5
diInstance = diDeviceType.getDeviceTypeInstance(5)

# Display the name of the DigitalInput instance in the UAB log file
(as a debug message)
thePlugin.writeDebugInUABLog(diInstance.getAttributeData("DeviceIdentifi
```

---

getDeviceTypeName

```
public java.lang.String getDeviceTypeName()
```

Method used to get the device type name.

Returns:

A String containing the device type name.

---

getObjectType

```
public java.lang.String getObjectType()
```

Method used to get the ObjectTypeFamily from the device type definition.

Returns:

A String containing the family type of the device type.

Currently, the available families are:

- IOObjectFamily
- InterfaceObjectFamily
- FieldObjectFamily
- ControlObjectFamily

---

getPackageName

`public java.lang.String` **`getPackageName`**`()`

Method used to get the name of the package which the device belongs to (CPC, CRYO, ...).

Returns:

A String containing the package name.

---

getSpecificationAttributes

`public java.util.List` **`getSpecificationAttributes`**`()`

Get the list of specification attributes (e.g.: DeviceIdentification:Name, DeviceIdentification:Expert Name, ...)

Returns:

- The list of specification attributes

Interface ISpecChange

< Methods >

---

public interface **ISpecChange**

Interface to read/write the specs change data.

Author:

Ivan Prieto Barreiro

Methods

getComments

`public java.lang.String` **`getComments`**`()`

Get the change comments.

Returns:

---

getDate

`public java.lang.String` **`getDate`**`()`

Get the date when the change was made.

Returns:

---

getHtmlLabel

`public java.lang.String` **`getHtmlLabel`**`()`

Get the Jira ticket ID.

Returns:

---

getHtmlLink

`public java.lang.String` **`getHtmlLink`**`()`

Get the Jira link.

Returns:

---

getUser

`public java.lang.String` **`getUser`**`()`

Get the alias of the user who made the change.

Returns:

---

getVersion

`public java.lang.String` **`getVersion`**`()`

Get the specs version after the change.

Returns:

---

setComments

```
public void setComments(java.lang.String comments)
```

Set the change comments.

Parameters:

comments -

---

setDate

```
public void setDate(java.lang.String date)
```

Set the date of the change.

Parameters:

date -

---

setHtmlLink

```
public void setHtmlLink(java.lang.String ticket,
                        java.lang.String link)
```

Set the Jira link.

Parameters:

ticket - Jira ticket ID.
link - Jira link.

---

setUser

```
public void setUser(java.lang.String user)
```

Set the alias of the user who made the change.

Parameters:

user - User alias.

---

setVersion

```
public void setVersion(java.lang.String version)
```

Set the specs version after the change.

Parameters:

version -

Interface ISpecDocumentation

< Methods >

public interface **ISpecDocumentation**

Interface to read/write from the Specs' ProjectDocumentation sheet.

Author:

Ivan Prieto Barreiro

Methods

addSpecChange

`public void` **`addSpecChange`**`(ISpecChange change)`

Add a new change to the specs.

Parameters:

change -

getApplicationName

`public java.lang.String` **`getApplicationName`**`()`

Get the application name.

Returns:

getFaqLink

`public java.lang.String` **`getFaqLink`**`()`

Get the FAQ URL.

Returns:

getObjectDescriptionsLink

`public java.lang.String` **`getObjectDescriptionsLink`**`()`

Get the object descriptions URL.

Returns:

---

getProjectDescription

`public java.lang.String` **`getProjectDescription`**`()`

Get the project description.

Returns:

---

getProjectName

`public java.lang.String` **`getProjectName`**`()`

Get the project name.

Returns:

---

getProjectOwner

`public java.lang.String` **`getProjectOwner`**`()`

Get the project owner alias.

Returns:

---

getSpecChanges

`public java.util.List` **`getSpecChanges`**`()`

Get the list of changes in the specs.

Returns:

---

getSpecsVersion

`public java.lang.String` **`getSpecsVersion`**`()`

Get the specs version.

Returns:

The version number of the last spec changes or the empty string if there are no changes.

---

newSpecChange

`public` `ISpecChange` **`newSpecChange`**`()`

Create a new specs change.

Returns:

---

setApplicationName

`public void` **`setApplicationName`**`(java.lang.String name)`

Set the application name.

Parameters:

name - The application name.

---

setFaqLink

`public void` **`setFaqLink`**`(java.lang.String faqLink)`

Set FAQ URL

Parameters:

faqLink - URL

---

setObjectDescriptionsLink

`public void` **`setObjectDescriptionsLink`**`(java.lang.String objectDescriptionsLink)`

Set object descriptions URL.

Parameters:

objectDescriptionsLink - url

---

setProjectDescription

`public void` **`setProjectDescription`**`(java.lang.String desc)`

Set the project description.

Parameters:

desc - The project description.

---

setProjectName

`public void` **`setProjectName`**`(java.lang.String name)`

Set the project name.

Parameters:

name - The project name.

---

setProjectOwner

`public void` **`setProjectOwner`**`(java.lang.String owner)`

Set the project owner alias.

Parameters:

owner - Project owner alias.

Interface ISpecFileTemplate

< Methods >

---

public interface **ISpecFileTemplate**

Interface used to get data from the specifications file.

The methods described in this interface can be used by the Jython template developers to complete the logic templates.

Note: The examples provided use the Jython syntax.

Author:

Ivan Prieto Barreiro

Methods

Dcount

```
public int Dcount(java.lang.String field,
                  java.lang.String type,
                  java.lang.String condition)
```

Count the number of elements found by the {@link #Dlookup(String, String, String)} method.

Iterates through all the instances of the device type specified. If the instance matches the specified condition it will add the value of the specified 'field' to the returned vector.

Parameters:

field - The field required for the returned String.
type - The device type name.
condition - The condition to match.

The binary operators available for the conditions are: "!=", "=", "contains", "startsWith", "endsWith", "matches"

It's possible to create complex conditions using the logical operators: 'and', 'or', 'not'.

Returns:

The number of elements found.

DependentLoop

```
public java.util.List DependentLoop(java.lang.String deviceTypeName,
                                     java.lang.String master,
                                     int start,
                                     int step,
                                     java.lang.String textRepeated)
```

Create a String vector with the 'textRepeated' for each device instance matching the specified conditions.

Parameters:

deviceTypeName - The device type name.
master - Value of the #LogicDeviceDefinitions:Master# field of the device in the spec file.
start - First value of the counter for the loop.
step - Amount counter is increased each time through the loop.
textRepeated - String expression to be repeated if the field Master in the Object equals the value Master.

Returns:

A string vector containing the 'textRepeated' for each instance matching the specified conditions.

Example: Display the name of all the instances of a device type with the specified master object

```
# For each AnalogAlarm instance which master object is
'DEMON_1_EH01AD', create a text displaying
# the counter value and the instance name.
text = theRawInstances.DependentLoop("AnalogAlarm",
"DEMON_1_EH01AD", 1, 1, "#counter# - #DeviceIdentification:Name#")

# Show the created text
for str in text:
    thePlugin.writeDebugInUABLog("$str$")
```

Output:

An example of the debug messages displayed in the user report window is:

1 - AnalogAlarm_1

2 - AnalogAlarm_2

...

DependentLoop

```
public java.util.List DependentLoop(java.lang.String deviceTypeName,
                                     java.lang.String master,
                                     int start,
                                     int step,
                                     java.lang.String textRepeated,
                                     java.lang.String condition)
```

Create a String vector with the 'textRepeated' for each device instance matching the specified conditions.

Parameters:

deviceTypeName - The device type name.
master - Value of the #LogicDeviceDefinitions:Master# field of the device in the spec file.
start - First value of the counter for the loop.
step - Amount counter is increased each time through the loop.
textRepeated - String expression to be repeated if the field Master in the Object equals the value Master.
condition - String expression evaluated for each step. If True, 'textRepeated' is append to the output vector.

The binary operators available for the conditions are: "!=", "=", "contains", "startsWith", "endsWith", "matches"

It's possible to create complex conditions using the logical operators: 'and', 'or', 'not'.

Returns:

A string vector containing the 'textRepeated' for each instance matching the specified conditions.

Example: Display the name of all the instances of a device type with the specified master object

```
# For each AnalogAlarm instance of type 'TS' which master object is
'DEMON_1_EH01AD', create a text displaying
# the counter value and the instance name.
text = theRawInstances.DependentLoop("AnalogAlarm",
"DEMON_1_EH01AD", 1, 1, "#counter# - #DeviceIdentification:Name#",
"'#FEDeviceAlarm:Type#'='TS'")

# Show the created text
for str in text:
    thePlugin.writeDebugInUABLog("$str$")
```

Output:

An example of the debug messages displayed in the user report window is:

1 - AnalogAlarm_1

2 - AnalogAlarm_2

...

DependentLoopString

```
public java.lang.String DependentLoopString(java.lang.String deviceTypeName,
                                             java.lang.String master,
                                             int start,
                                             int step,
                                             java.lang.String textRepeated)
```

Similar to {@link #DependentLoop(String, String, int, int, String)}.

This method returns only one String where the different lines are separated by end-of-line char ("\n").

Parameters:

deviceTypeName - The device type name.
master - Value of the #LogicDeviceDefinitions:Master# field of the device in the spec file.
start - First value of the loop.
step - Amount counter is increased each time through the loop.
textRepeated - String expression to be repeated if the field Master in the Object equals the value Master.

Returns:

A string containing the 'textRepeated' for each instance matching the specified conditions.

DependentLoopString

```
public java.lang.String DependentLoopString(java.lang.String deviceTypeName,
                                             java.lang.String master,
                                             int start,
                                             int step,
                                             java.lang.String textRepeated,
                                             java.lang.String condition)
```

Similar to {@link #DependentLoop(String, String, int, int, String, String)}

This method returns only one String where the different lines are separated by end-of-line char ("\n").

Parameters:

deviceTypeName - The device type name.
master - Value of the #LogicDeviceDefinitions:Master# field of the device in the spec file.
start - First value of the loop.
step - Amount counter is increased each time through the loop.
textRepeated - String expression to be repeated if the field Master in the Object equals the value Master.
condition - String expression evaluated for each step. If True, TextRepeated is append to the output vector. The binary operators available for the conditions are: "!=", "=", "contains", "startsWith", "endsWith", "matches"

It's possible to create complex conditions using the logical operators: 'and', 'or', 'not'.

Returns:

A string containing the 'textRepeated' for each instance matching the specified conditions.

Dlookup

```
public java.util.List Dlookup(java.lang.String field,
                              java.lang.String type,
                              java.lang.String condition)
```

Dependent look up method. Iterates through all the instances of the device type specified. If the instance matches the specified criteria it will add the value of the specified 'field' to the returned vector.

Parameters:

field - The field required for the returned String.
type - The device type name.
condition - The condition to match.

The binary operators available for the conditions are: "!=", "=", "contains", "startsWith", "endsWith", "matches"

It's possible to create complex conditions using the logical operators: 'and', 'or', 'not'.

Returns:

A String vector containing the value of the 'field' attribute for all the instances matching the specified criteria. If there are no instances matching the criteria an empty vector will be returned

Example: Get the name of all the Analog Alarms which type is 'AL'

```
result = theRawInstances.Dlookup("DeviceIdentification:Name",
"AnalogAlarm", "'#FEDeviceAlarm:Type#'='AL'")

# Display the name of all the analog alarms matching the criteria
in the UAB log (as debug message)
for alarmName in result:
    thePlugin.writeDebugInUABLog("Analog Alarm: $alarmName$")
```

DlookupString

```
public java.lang.String DlookupString(java.lang.String field,
                                       java.lang.String type,
                                       java.lang.String condition)
```

Dependent look up String method. Similar to the Dlookup method, but in this case the returned value will be the 'field' value of the **first instance** matching the criteria.

Parameters:

field - The field required for the returned String.
type - The device type name.
condition - The condition to match.

The binary operators available for the conditions are: "!=", "=", "contains", "startsWith", "endsWith", "matches"

It's possible to create complex conditions using the logical operators: 'and', 'or', 'not'.

Returns:

A String containing the value of the 'field' attribute for all the instances matching the specified criteria. If there are no instances matching the criteria it will return the empty String. {@code}

Example: Get the name of all the Analog Alarms which type is 'AL'

```
result = theRawInstances.DlookupString("DeviceIdentification:Name",
"AnalogAlarm", "'#FEDeviceAlarm:Type#'='AL'")

# Display the name of all the first analog alarm matching the
criteria in the UAB log (as debug message)
thePlugin.writeDebugInUABLog("Analog Alarm: $result$")
```

GenericDepLoop

```
public java.lang.String GenericDepLoop(java.lang.String instanceName,
                                       java.lang.String targetDevice,
                                       java.lang.String linkedField,
                                       int optionDuplicate,
                                       int first,
                                       int fStep,
                                       java.lang.String fTextWithLink,
                                       java.lang.String fTextNoLink)
```

This function detects link between an object name in another object table (targetDevice) in a particular field (LinkedField).

If a link is detected then it will replace in the string FTextWithLink the fields between # with the correspondent value in the target device and returns the string to the function.

If several links are found, there are three different options:
- optionDuplicate=0 then it will concatenate the several strings;
- optionDuplicate=1 then it will keep the first replaced string;
- optionDuplicate=2 then it will keep the last replaced string;

If no link is detected then, it will return the string FTextNoLink.

Parameters:

instanceName - Name of the device instance to look for.
targetDevice - Name of the device type where to look for the link.
linkedField - Field where to look for the specified link.
optionDuplicate - Integer value to specify if the generated text must be duplicated for each device instance. The possible values are:
- **0:** The fTextWithLink String will be concatenated for each link detected.
- **1:** The fTextWithLink String will be generated **only** for the **first link** detected.
- **2:** The fTextWithLink String will be generated **only** for the **last link** detected.

first - Integer used as the value for the first instance.
fStep - Integer used as step to increment the 'first' value of the previous instance.
fTextWithLink - Text to return if there are one or more links found.
fTextNoLink - Text to return it there are no links found.

Returns:

A string with the generated text.

Example: Display the name of the AnalogAlarm instances which #FEDeviceManualRequests:LL Alarm# field in the spec file contains the value 'AnalogParameter_1'

```
# optionDuplicate = 0
generatedText0 =
theRawInstances.GenericDepLoop("AnalogParameter_1", "AnalogAlarm",
"FEDeviceManualRequests:LL Alarm", 0, 1, 1, "#counter# -
#DeviceIdentification:Name#", "No links found!")

# optionDuplicate = 1
generatedText1 =
theRawInstances.GenericDepLoop("AnalogParameter_1", "AnalogAlarm",
"FEDeviceManualRequests:LL Alarm", 1, 1, 1, "#counter# -
#DeviceIdentification:Name#", "No links found!")

# optionDuplicate = 2
```

```
generatedText2 =
theRawInstances.GenericDepLoop("AnalogParameter_1", "AnalogAlarm",
"FEDeviceManualRequests:LL Alarm", 2, 1, 1, "#counter# -
#DeviceIdentification:Name#", "No links found!")
```

Output:

If the specs file contains, for example, three instances of the AnalogAlarm device type matching the specified conditions, the content of the generatedText variables will be the following:

generatedText0 variable (with optionDuplicate = 0)

1 - AnalogAlarm_1

2 - AnalogAlarm_2

3 - AnalogAlarm_3

generatedText1 variable (with optionDuplicate = 1)

1 - AnalogAlarm_1

generatedText2 variable (with optionDuplicate = 2)

3 - AnalogAlarm_3

Note: If there are no AnalogAlarm instances matching the specified conditions the content of the three generatedText variables will be the String: "No links found!"

createSectionText

```
public java.lang.String createSectionText(java.util.List theTypeInstances,
                                          int optionDuplicate,
                                          int counterStart,
                                          int counterStep,
                                          java.lang.String textIfLink,
                                          java.lang.String textIfNoLink)
```

This method is used to generate text for a program/section.

Parameters:

theTypeInstances - Vector with all the necessary instances to produce the output text.
optionDuplicate - Integer value to specify if the generated text must be duplicated for each device instance. The possible values are:

- **0:** The textIfLink String will be generated for each instance in theTypeInstances vector.
- **1:** The textIfLink String will be generated **only** for the **first instance** of theTypeInstances vector.
- **2:** The textIfLink String will be generated **only** for the **last instance** of theTypeInstances vector.

counterStart - Integer used as the value for the first instance.
counterStep - Integer used as step to increment the 'counterStart' of the previous object.
textIfLink - Text to be inserted for each instance in the vector. It's possible to perform string replacements using the specs path of the required data (see the example below).
textIfNoLink - If the instances vector provided doesn't have any instance, the String contained in this parameter will be returned.

Returns:

A string with the generated text.

Example:

```
# Get the DigitalInput device type
diDeviceType = theRawInstances.getDeviceType("DigitalInput")

# Get a vector with all the DigitalInput instances
diInstances = diDeviceType.getAllDeviceTypeInstances()

# optionDuplicate = 0
generatedText0 = theRawInstances.createSectionText(diInstances, 0,
1, 1, "#counter# - Device name: #DeviceIdentification:Name#", "No
instances provided!")

# optionDuplicate = 1
generatedText1 = theRawInstances.createSectionText(diInstances, 1,
1, 1, "#counter# - Device name: #DeviceIdentification:Name#", "No
instances provided!")

# optionDuplicate = 2
generatedText2 = theRawInstances.createSectionText(diInstances, 2,
1, 1, "#counter# - Device name: #DeviceIdentification:Name#", "No
instances provided!")
```

Output:

If the specs file contains, for example, three instances of the DigitalInput device type the content of the generatedText variables will be the following:

generatedText0 variable (with optionDuplicate = 0)

1 - Device name: DigitalInput_1

2 - Device name: DigitalInput_2

3 - Device name: DigitalInput_3

generatedText1 variable (with optionDuplicate = 1)

1 - Device name: DigitalInput_1

generatedText2 variable (with optionDuplicate = 2)

3 - Device name: DigitalInput_3

Note: If there are no DigitalInput instances defined in the spec file, the content of the three generatedText variables will be the String: "No instances provided!"

createSectionText

```
public java.lang.String createSectionText(java.util.List theTypeInstances,
                                          int counterStart,
                                          int counterStep,
                                          java.lang.String textRepeated)
```

This method is used to generate text for a program/section.

Parameters:

theTypeInstances - Vector with all the necessary instances to produce the output text.
counterStart - Integer used as the value for the first instance.
counterStep - Integer used as step to increment the 'counterStart' of the previous object.
textRepeated - Text to be inserted for each instance in the vector. It's possible to perform string replacements using the specs path of the required data (see the example below).

Returns:

A string with the generated text.

Example:

```
# Get the DigitalInput device type
diDeviceType = theRawInstances.getDeviceType("DigitalInput")

# Get a vector with all the DigitalInput instances
diInstances = diDeviceType.getAllDeviceTypeInstances()

generatedText = theRawInstances.createSectionText(diInstances, 1,
1, "#counter# - Device name: #DeviceIdentification:Name#")
```

Output:

The example above will store in the generatedText variable a string containing, for each instance of the DigitalInput device type, the instance number and its name as specified in the specs file, e.g.:

1 - Device name: DigitalInput_1

2 - Device name: DigitalInput_2

...

---

findInstanceByName

```
public research.ch.cern.unicos.utilities.IDeviceInstance
findInstanceByName(java.lang.String instanceName)
```

Find a device instance by name.

Parameters:

instanceName - The name of the instance to look for.

Returns:

The IDeviceInstance found or null if it doesn't exist.

findInstanceByName

```
public research.ch.cern.unicos.utilities.IDeviceInstance
findInstanceByName(java.lang.String deviceTypeNames,
java.lang.String instanceName)
```

Find a device instance by name.

Parameters:

deviceTypeNames - Device type name(s) where the method will look for the instances (e.g.: "DigitalInput,DigitalOutput"). The string "*" can be used to look in all the device types.
instanceName - The name of the instance to look for.

Returns:

The IDeviceInstance found or null if it doesn't exist.

findInstanceByNameIgnoreCase

```
public research.ch.cern.unicos.utilities.IDeviceInstance
findInstanceByNameIgnoreCase(java.lang.String instanceName)
```

Find a device instance by name, case insensitive.

Parameters:

instanceName - The name of the instance to look for.

Returns:

The IDeviceInstance found or null if it doesn't exist.

findInstanceByNameOrExpertName

```
public research.ch.cern.unicos.utilities.IDeviceInstance
findInstanceByNameOrExpertName(java.lang.String instanceName,
java.lang.String deviceTypeNames)
```

Find a device instance by name or expert name. Expert name take precedence.

Parameters:

instanceName - The name of the instance to look for.
deviceTypeNames - name of the device type or comma separated list of device types to look for given instance

Returns:

The IDeviceInstance found or null if it doesn't exist.

findMatchingInstances

```
public java.util.List findMatchingInstances(java.lang.String deviceTypeNames,
                                            java.lang.String condition)
```

This method is used to get all the instances of the specified device type(s) where the specified condition is true.

Parameters:

deviceTypeNames - Device type name(s) where the method will look for the instances (e.g.: "DigitalInput,DigitalOutput"). The string "*" can be used to look in all the device types. condition - The condition to be fulfilled by the instances (e.g.: '"#LogicDeviceDefinitions:Type#'='FS'")

It's possible to check several conditions using the "," separator (interpreted as the AND operator).

e.g. ('"#LogicDeviceDefinitions:Type#'='FS','#DeviceIdentification:Name#'='Instance1'") will look for the instance with name 'Instance1' which LogicDeviceDefinition:Type field in the spec file is 'FS'.

The binary operators available for the conditions are: "!=", "=", "contains", "startsWith", "endsWith", "matches"

It's possible to create complex conditions using the logical operators: 'and', 'or', 'not'.

Returns:

A vector with all the instances fulfilling the specified condition.

Example: Get the name of all the Analog Alarm devices which type is 'AL'

```
alarms = theRawInstances.findMatchingInstances("AnalogAlarm",
"'#FEDeviceAlarm:Type#'='AL'")

# Display the name of all the devices found in the UAB log (as
debug message)
for alarm in alarms:
    thePlugin.writeDebugInUABLog("Analog Alarm: " +
alarm.getAttributeData("DeviceIdentification:Name"))
```

findMatchingInstances

```
public java.util.List findMatchingInstances(java.lang.String deviceTypeNames,
                                            java.lang.String masterObject,
                                            java.lang.String condition)
```

This method is used to get all the instances of an specified device type with an specified master object, where the specified condition is true.

Parameters:

deviceTypeNames - Device type name(s) where the method will look for the instances (e.g.: "DigitalInput,DigitalOutput"). The string "*" can be used to look in all the device types.
masterObject - Name of the required master object for the instances.
condition - The condition to be fulfilled by the instances (e.g.: '"#LogicDeviceDefinitions:Type#'='FS'")

It's possible to check several conditions using the "," separator (interpreted as the AND operator).

e.g. ('"#LogicDeviceDefinitions:Type#'='FS','#DeviceIdentification:Name#'='Instance1'") will look for the instance with name 'Instance1' which LogicDeviceDefinition:Type is 'FS'.

The binary operators available for the conditions are: "!=", "=", "contains", "startsWith", "endsWith", "matches"

It's possible to create complex conditions using the logical operators: 'and', 'or', 'not'.

Returns:

A vector with all the instances fulfilling the commented conditions

Example: Get the name of all the Analog Alarm devices which type is 'AL'

```
alarms = theRawInstances.findMatchingInstances("AnalogAlarm",
"DEMON_1_EH01AD", "'#FEDeviceAlarm:Type#'='AL'")

# Display the name of all the devices found in the UAB log (as
debug message)
for alarm in alarms:
    thePlugin.writeDebugInUABLog("Analog Alarm: " +
alarm.getAttributeData("DeviceIdentification:Name"))
```

findMatchingInstances

```
public java.util.List findMatchingInstances(java.lang.String deviceTypeNames,
                                            java.lang.String masterObject,
                                            java.lang.String condition,
                                            java.util.List positions)
```

This method is used to get all the instances of an specified device type with an specified master object, where the specified condition is true.

Parameters:

deviceTypeNames - Device type name(s) where the method will look for the instances (e.g.: "DigitalInput,DigitalOutput"). The string "*" can be used to look in all the device types.
masterObject - Name of the required master object for the instances.
condition - The condition to be fulfilled by the instances (e.g.: '"#LogicDeviceDefinitions:Type#'='FS'")

It's possible to check several conditions using the "," separator (interpreted as the AND operator).

e.g. ('"#LogicDeviceDefinitions:Type#'='FS','#DeviceIdentification:Name#'='Instance1'") will look for the instance with name 'Instance1' which LogicDeviceDefinition:Type is 'FS'.

The binary operators available for the conditions are: "!=", "=", "contains", "startsWith", "endsWith", "matches"

It's possible to create complex conditions using the logical operators: 'and', 'or', 'not'.
positions - For each device found, the list will contain the position where the 'masterObject' is found, e.g. if the specs field 'LogicDeviceDefinitions:Master' contains the values 'PCO, PCO1, PCO2', and the parameter 'masterObject' contains the value 'PCO', then the list will contain the position index '0' for that instance.

Returns:

A vector with all the instances fulfilling the specified condition(s).

Example: Get the name of all the Analog Alarm devices which type is 'Multiple' and which master object is 'DEMON_1_CV01Ana'

```
# Create a list to get the master object positions
positions = ArrayList()

# Get the required instances
alarms = theRawInstances.findMatchingInstances("AnalogAlarm",
"DEMON_1_CV01Ana", "'#FEDeviceAlarm:Type#'='Multiple'", positions)

# Display the name of all the devices found and the master's
position in the UAB log (as debug message)
i = 0
for alarm in alarms:
    thePlugin.writeDebugInUABLog("Analog Alarm: " +
alarm.getAttributeData("DeviceIdentification:Name"))
    thePlugin.writeDebugInUABLog("Master position: " +
str(positions.get(i)))
    i = i+1
```

getAllDeviceTypes

`public java.util.List` **`getAllDeviceTypes`**`()`

Get all the device types available in the specs file.

Returns:

A vector with all the existing device type interfaces.

Example: Get the description of all the device types available in the specs file.

```
deviceTypes = theRawInstances.getAllDeviceTypes()

# Display the description of all the device types in the UAB log
(as debug message)
for deviceType in deviceTypes:
    thePlugin.writeDebugInUABLog("Device Type: " +
deviceType.getDeviceTypeName() + " Description: " +
deviceType.getDescription())
```

getAllDeviceTypesString

`public java.lang.String` **`getAllDeviceTypesString`**`()`

Get a String containing all the device type names separated by the character ','.

Returns:

A String containing all the device type names separated by the character ','.

Example: Get a string containing all the device types available in the specs (comma separated).

```
deviceTypes = theRawInstances.getAllDeviceTypesString()

# Show the device types string in the UAB log (as debug message)
thePlugin.writeDebugInUABLog("Device Types: $deviceTypes$")
```

getDeviceType

```
public research.ch.cern.unicos.utilities.IDeviceType
getDeviceType(java.lang.String theDeviceTypeName)
```

Get a device type from the specs file.

Parameters:

theDeviceTypeName - The device type name (e.g. "DigitalInput")

Returns:

The device type interface if exists, otherwise null.

Example: Get the description of the DigitalInput device type.

```
deviceType = theRawInstances.getDeviceType("DigitalInput")

# Display the description of the device type in the UAB log (as
debug message)
thePlugin.writeDebugInUABLog("Device Type: " +
deviceType.getDeviceTypeName() + " Description: " +
deviceType.getDescription())
```

---

getProjectDocumentation

```
public ISpecDocumentation getProjectDocumentation()
```

Get the project documentation data from the specs file.

Returns:

The specs documentation interface to handle the Project Documentation data.

---

getResourcesName

```
public java.lang.String getResourcesName()
```

Get the name of the resources package used to build the specs file.

Returns:

A string containing the name of the resources package used to build the specs file. (e.g. "cpc-resources-package").

---

getResourcesVersion

`public java.lang.String` **`getResourcesVersion`**`()`

Get the version of the resources package used to build the specs file.

Returns:

A String containing the version of the resources package used to build the specs file. (e.g. "1.3.0")

---

getSpecsPath

`public java.lang.String` **`getSpecsPath`**`()`

Return the path to the specs file.

Returns:

INDEX

I

N

S

W