# Asyril

# Asycube Provider

Version 1.1.0

# User’s guide

May 12, 2017

NOTES:

## [ Revision history ]

| Version | Date | Contents |
|---|---|---|
| 1.0.0 | 2016-01-26 | First edition. |
| 1.1.0 | 2017-05-12 | Add send raw command and execute sequence |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Contents

# 1. Introduction

This document is a user's guide of the Asycube provider that is the CAO provider for the feeder systems manufactured by Asyril.

The Asycube provider connects with the feeder with ethernet **TCP/IP messaging only**, managing all low level communication. Feeder with a serial interface interface (RS232/RS485) needs a serial to Ethernet converter.

# 2. Outline of provider

## 2.1. Outline

This provider gives a straightforward access to the base control commands of an Asycube feeder.

The three main parts of the feeder, the bulk, the plate and the backlight can be addressed through commands using the CaoController::Execute() call.

To configure and tune the vibration settings use the Asyril HMI software.

**Table 2-1 Asyril Asycube provider**

| | |
|---|---|
| File name | asycubeDIL.dll |
| ProgID | CaoProv.Asyril.Asycube |
| Registry registration | regsvr32 asycubeDIL.dll |
| Registry un-registration | regsvr32 /u asycubeDIL.dll |

## 2.2. Feeder minimum firmware requirement

This provider requires an Asycube feeder with the following minimum firmware requirement

Asycube model 240. Firmware: V 2.2.0

Asycube models 50 and 80. Firmware V3.0.0

## 2.3. Method and property

### 2.3.1. CaoWorkspace::AddController method

Syntax

AddController ( <CtrlName>, <ProvName >,<ExecMachineName>,<OptionsStr)

| | |
|---|---|
| CtrlName | : [in] Controller name. Arbitrary string. (Ex. "Feeder") |
| ProvName | : [in] Provider name. (Fixed to "**CaoProv.Asyril.Asycube**") |
| ExecMachineName | : [in] Execution machine name of provider. **Not used. Leave empty**. |
| OptionStr | : [in] Option character string |

| **Option** | **Meaning** |
|---|---|
| Conn=<connection parameter> | Set the communication parameter.<br>The only valid communication protocol valid at this time for this provider = TCP. |
| Timeout=<Delay> | Set the TCP socket communication timeout. Unit [ms]<br>A timeout error will occurs if the TCP connection is not established after that delay. |
| NodeNo=<Address number> | Mandatory only for the RS232/485 feeder model with a physical rotative switch at the back of the device.<br>The NodeNo parameter must match the rotative switch selection.<br>Integer value. Range [0-15]<br>For the Ethernet only Asycube this parameter can be omitted. |
| Mode=<RawComChannel> | Not detailed here. For Asyril use only. |
| Descriptor=<Descriptor chain> | Not detailed here. For Asyril use only. |

**Examples**

Typical for RS232/485 Asycubes (product names "mezzo", "forte")

```
cao.AddController("Feeder1", "CaoProv.Asyril.Asycube", "",
"conn=TCP:192.168.127.253:4001,Timeout=2000,NodeNo=1")
```

Typical for Ethernet only Asycube (product name "largo", "50","80", "240")

```
cao.AddController("Feeder1", "CaoProv.Asyril.Asycube", "",
"conn=TCP:192.168.127.253:4001,Timeout=2000")
```

# 3. Command reference

## 3.1. CaoController::Execute(“<Command name>”) command

Syntax <Command>,<Value>

Note: The vibrations settings must be tuned using the Asyril HMI interface.

The feeder can be easily controlled by the control commands using the standard direction keyword specified in the Asycube documentation.

The Figure 1 below summarizes the basic directions with the associated control command. The table 1 list all control commands. For advance use see Annex I "List of configuration & management commands".

Note on the command syntax: **The command string is not case-sensitive.**

Note on the script execution: **By default all commands will block the script execution (synchronous call) until the action is done**. However some of the command allow to use the **"!" sign** that will make the same command execute without blocking (asynchronous call). E.g. "**!**Move.Forward".

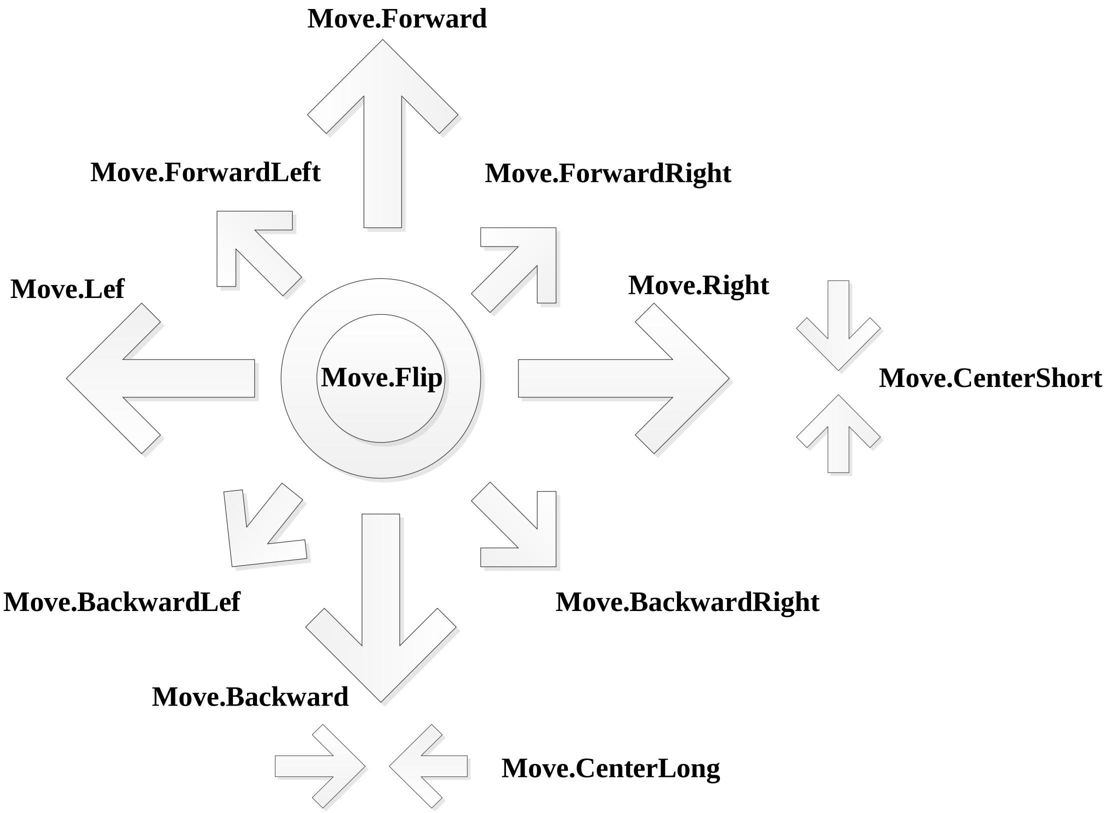

Fig 1. Move direction with corresponding commands

## 3.2. List of control commands

<table>
<tr><th>#</th><th>&lt;Command&gt;</th><th>&lt;Value&gt;</th><th>Description</th></tr>
<tr><td><b>1</b></td><td>"ComChannel.Send"</td><td>TextCmd<br>[String]</td><td>Sending a "raw" text command to the feeder.<br>Returning the response as a raw String.<br>Using the feeder specific protocol syntax.<br><br>Examples :<br><br>To select a vibration batch :<br><i>"ComChannel.Send","{UV1}"</i><br>Returns<br><i>"{UV01}"</i><br><br>To execute a sequence :<br><i>"ComChannel.Send",</i><br><i>"{ES:(NbParts;NbPartsMax;XCenter;YCenter;SeqID}"</i><br>Returns<br><i>"{ES:(NbParts;NbPartsMax;XCenter;YCenter;SeqID;Duration)}"</i><br><br>See the "Programming Manual" of your Asycube for more details on the syntax.<br><br>This command is never blocking the script. If a vibration with a given duration is called, you have to extract manually the duration and wait for the end of the vibration.</td></tr>
<tr><td><b>2</b></td><td>"<i>[!]</i>Feed.&lt;<i>FeedDirection</i>&gt;"</td><td>Duration [ms]<br>[1]</td><td>For Asycube with integrated bulk only.<br>Vibrate the bulk for the given duration [ms] in the direction specified by the &lt;FeedDirection&gt; keyword.<br><table><tr><th>#</th><th><i>&lt;FeedDirection&gt;</i></th></tr><tr><td><b>1</b></td><td><i>"Forward"</i></td></tr><tr><td><b>2</b></td><td><i>"Backward"</i></td></tr></table><br>Eg. <i>"Feed.Forward"</i></td></tr>
<tr><td><b>3</b></td><td>"Feed.ReadState"</td><td>-</td><td>Reading the current bulk state<br>The returned value in an integer that code for 4 different states.<br><table><tr><th>State value</th><th>Vibration</th></tr><tr><td>0</td><td>Disable</td></tr><tr><td>1</td><td>Stopped</td></tr><tr><td>3</td><td>Vibrating</td></tr><tr><td>5</td><td>Overheat</td></tr></table><br>See the "Programming manual" for more information.</td></tr>
<tr><td><b>5</b></td><td>"Feed.Stop"</td><td>-</td><td>Stop immediately the current bulk vibration.</td></tr>
</table>

<table>
<tr><td><b>5</b></td><td>"<i>[!]</i>Move.&lt;<i>MoveDirection</i>&gt;"</td><td>Duration [ms]<br>[1]</td><td>Vibrate the platform for the given duration [ms] in the direction specified by the &lt;<i>MoveDirection</i>&gt; keyword.<br><table><tr><th>#</th><th><i>&lt;MoveDirection&gt;</i></th></tr><tr><td><b>1</b></td><td><i>"Forward"</i></td></tr><tr><td><b>2</b></td><td><i>"Left"</i></td></tr><tr><td><b>3</b></td><td><i>"Right"</i></td></tr><tr><td><b>4</b></td><td><i>"ForwardLeft"</i></td></tr><tr><td><b>5</b></td><td><i>"ForwardRight"</i></td></tr><tr><td><b>6</b></td><td><i>"Backward"</i></td></tr><tr><td><b>7</b></td><td><i>"BackwardLeft"</i></td></tr><tr><td><b>8</b></td><td><i>"BackwardRight"</i></td></tr><tr><td><b>9</b></td><td><i>"Flip"</i></td></tr><tr><td><b>10</b></td><td><i>"CenterShort"</i></td></tr><tr><td><b>11</b></td><td><i>"CenterLong"</i></td></tr></table><br>E.g. <i>"Move.Forward"</i></td></tr>
<tr><td><b>6</b></td><td>"Move.ReadState"</td><td>-</td><td>Reading the current platform state<br>The returned value in an integer that code for 4 different states.<br><table><tr><th>State value</th><th>Vibration</th></tr><tr><td>0</td><td>Disable</td></tr><tr><td>1</td><td>Stopped</td></tr><tr><td>3</td><td>Vibrating</td></tr><tr><td>5</td><td>Overheat</td></tr></table><br>See the "Programming manual" for more information.</td></tr>
<tr><td><b>7</b></td><td>"Move.Stop"</td><td>-</td><td>Stop immediately the current platform vibration.</td></tr>
<tr><td><b>8</b></td><td>"Backlight.Flash.Set"</td><td>Duration [ms]</td><td>Set the Asycube Backlight flash duration in [ms].<br>Note: There is no need to set the flash duration for every cycle. The value is maintained into the RAM. For better timing performance, use the Flash.Set only when the flash duration needs to be changed.</td></tr>
<tr><td><b>9</b></td><td>"<i>[!]</i>Backlight.Flash"</td><td>-</td><td>Trig a backlight flash for the duration previously set with the "Backlight.Flash.Set" command.</td></tr>
<tr><td><b>10</b></td><td>"Backlight.Set"</td><td>[True/False]</td><td>Turning the backlight on or off permanently.<br><br>Note: The backlight will turn off automatically to protect from overheat if it's activated for a too long time.<br>See the "ClearAlarms" command to reactivate the backlight.<br>See the "Programming manual" for more details.</td></tr>
</table>

<table>
<tr><td><b>11</b></td><td>"Backlight.ReadState"</td><td>-</td><td>Return the activation state of the backlight.<br>The return value is an integer that code for the current state.<br>0 : → The backlight is OFF<br>1: → The backlight is ON</td></tr>
<tr><td><b>12</b></td><td>"Outputs[<OutNo>].Analog[<AnalogNo>].Set"</td><td>Output level [%]</td><td>For Asycube with integrated outputs only.<br>Setting the analog output level (nbr 1 or 2) that will be applied when starting the outputs.<br>See the Output[<OutNo>].Start command.<br>See the "Programming manual" for more details</td></tr>
<tr><td><b>13</b></td><td>"<i>[!]</i>Outputs[<OutNo>].Start"</td><td>Duration [ms]<br>[1]</td><td>For Asycube with integrated outputs only.<br>Starting the standard output nbr 1 or 2 for the given duration [ms].<br>Note: The two digital outputs and the two analog outputs will be activated according to the configuration.<br>See the "Programming manual" for more details</td></tr>
<tr><td><b>14</b></td><td>"Outputs.ReadState"</td><td>-</td><td>Reading the current platform state<br>The returned value in an integer that code for 3 different states.
<table>
<tr><th>State value</th><th>Output</th></tr>
<tr><td>0</td><td>Disable</td></tr>
<tr><td>1</td><td>Stopped</td></tr>
<tr><td>3</td><td>Activating</td></tr>
</table>
See the "Programming manual" for more details</td></tr>
<tr><td><b>15</b></td><td>"Outputs.Stop"</td><td>-</td><td>For Asycube with integrated outputs only.<br>Stopping all activated outputs.<br>Note: Can be used to stop an output that was activated continuously.</td></tr>
<tr><td><b>16</b></td><td>"Din[<DinNo>].Read"</td><td>-</td><td>For Asycube with integrated outputs only.<br>Reading the state of one of the two digital inputs.<br>Returns an integer value.<br>0 → False,    1 → True</td></tr>
</table>

| | | | |
|---|---|---|---|
| **17** | "ReadWarning" | - | Return the value of the warning register.<br>A return value of "0" indicates no warning.<br>The returned code depends on the feeder model.<br><br>Note[1]: The returned code is a register of length 8 represented as an integer. This value code for up to 8 independent alarms. See the programming manual for more details.<br>See the command "ClearWarnings"<br><br>Note[2]: With the "Largo" product the two digital inputs state are returned with the bit 4 and 5. |
| **18** | "ClearWarnings" | - | Clears the warnings register. |
| **19** | "ReadAlarms" | - | Return the value of the alarms register.<br>A return value of "0" indicates no alarms.<br>An alarm will put the feeder in an alarm state. See the command "ClearAlarms" to reset.<br><br>Note: The returned code is a register of length 8 represented as an integer value. This value code for up to 8 independent alarms.<br>See the programming manual for more details. |
| **20** | "ClearAlarms" | - | Clears the alarms register |

Table 1. List of commands

**[1]** The duration argument is interpreted the following way.
a) Integer number > 0 → the value given as argument is applied
b) Integer number = 0 → the vibration is continuous (infinite duration). See "Move.Stop" command
c) Empty string "" → the duration stored into the corresponding batch is applied.
E.g. feeder.Execute("Move.Forward",150) . Vibrate for 150[ms]
feeder.Execute("Move.Forward",0) . Vibrate continuously until a "Move.Stop" is done.
feeder.Execute("Move.Forward","") .Vibrate for the duration stored into the batch "a"

*[!]* *C*ommand that accept the optional "!" prefix that makes an unblocking call (asynchronous)

## 3.3. Error messages

The following table details the error messages defined by the provider.

| # | Id | Symbol String | Message | Context | Typical cause |
|---|---|---|---|---|---|
| **#1** | 1 | NODE_ADDR_OUT_OF_RANGE | "Node address out of range" | When calling cao.AddController(…) | - The "NodeNo" argument of the AddController is out of the range [0-15] |
| **#2** | 2 | TCP_CONNEXION_TIMEOUT | "TCP connexion timeout" | When calling cao.AddController(…)<br>The tcp connexion could not be established within the "timeout" delay specified as AddController argument | - Ethernet not pluged (physically)<br>TCP/IP network parameter does not match<br>Timeout parameter is too low. (Typical connexion duration need approximately 10[ms]) |
| **#3** | 3 | TCP_SEND_ERROR | "TCP send error" | When calling any provider "Execute" that use internally communication.<br>Generated by the Socket->send() call | - TCP socket error |
| **#4** | 4 | RESPONSE_TIMEOUT | "Response timeout" | When calling any provider "Execute" function that make use of communication. | - Lost Eth connection<br>- Lost TCP connection<br>- Wrong "NodeNo" value that cause the device to reject silently any commands.<br>- Device not answering (for ex. powered-off) |
| **#5** | 5 | CMD_NOT_FOUND | "Command not found" | When calling the provider "Execute" function. Generated by the AsycubeModel->ExecuteCommand() | If the command character string is not a valid keyword.<br>Rem. The command string is not sensitive to the case. |
| **#6** | 6 | INVALID_CMD_ARGUMENT | "Invalid command argument" | When calling the provider "Execute" function. Generated by the ReadRegister or WriteRegister command | If the register number is not even. |
| **#7** | 7 | INVALID_BATCH_IDENTIFIER | "Invalid command identifier" | When calling the provider "Execute" function. When using a command that takes a batch identifier argument. | If the batch letter is not a valid identifier.<br>Valid batch identifier [A…Z] |

Table 2. Error messages

# 4. Sample program

This example shows a Denso PacScript that illustrate a feeding/spreading sequence pattern
This states that the Asycube is already configured to feed a particular part. Asyril's user interface must be used separately to tune all vibration settings.

The example uses an optimized feeding sequence that takes the **part mean location** to choose the appropriate spreading sequence.
Spreading is done by doing a **two steps vibration sequence. 1) Center parts 2) Flip parts**
The centering direction depends on the parts mean location compare to a 9 zone subdivision of the platform.
The timing depends on the distance to go to the platform center and can be computed with a linear relation.
This calculus is not done on this example for the sake of simplicity.

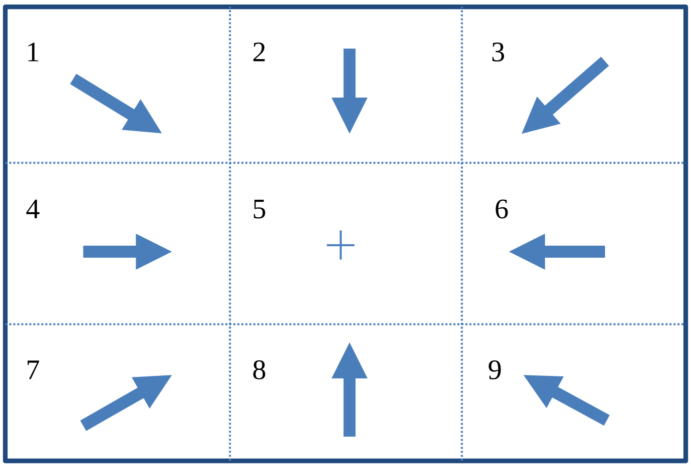

Fig 2. Centering direction depending on the parts mean location

| **Zone number** | **Spreading sequence** |
|---|---|
| **1** | ForwardRight + Flip |
| **2** | Right + Flip |
| **3** | BackwardRight + Flip |
| **4** | Forward + Flip |
| **5** | Flip |
| **6** | Backward + Flip |
| **7** | ForwardLeft + Flip |
| **8** | Left + Flip |
| **9** | BackwardLeft + Flip |

Table 3. Spreading sequence depending on the part location

```
'!TITLE "ACube_SampleCode"

' Author: Asyril
' This program shows the typical commands to pilot a Asycube.
' The provider works exclusively with TCP/IP communication. For the Asycube models that integrate only a RS232/485 interface,
' a TCP/IP <-> RS232/485 converter is required
' **************************************************************************************************************
' The Asycube is not configured via the script and must be tuned for vibration settings using the Asyril's user interface .
' **************************************************************************************************************

Sub Main

    Dim feeder as Object
    On Error Goto ErrorMgmt

    ' Creating   an instance of the Asycube controller
    ' Creating a feeder object for a feeder that integrates TCP/IP

    feeder = cao.AddController("Feeder1", "CaoProv.Asyril.Asycube", "", "conn=TCP:192.168.127.254:4001,Timeout=10000")

    Dim partNbr as Integer = 0
    Dim feedLimit as Integer = 5
    Dim partLocZone as Integer = 0
    Dim mvCenterDuration as Integer = 0

    '*****************************************************************************
    ' Insert here the dialog with vision to get the number and localization of parts
    ' partNbr = VisionSystem.QueryPart(...)
    ' partLocZone = VisionSystem.QueryPartLocZone(...)
    ' mvCenterDuration = VisionSystem.QueryCenterDistanceDelay(...)
    ' To synchronize the backlight with the camera use the "BackLight.Flash" command
    ' call feeder.Execute("!Backlight.Flash","") or use the digital input "synchroBackLight"
    '*****************************************************************************

    'Feeding more part if necessary
    If partNbr < feedLimit Then
        call feeder.Execute("Feed.Forward",1000)
    End if

    ' Spreading parts ont the platform by choosing the appropriate vibrations depending on the part mean localization zone

    Select Case partLocZone
        Case 1
            call feeder.Execute("Move.ForwardRight",mvCenterDuration)
        Case 2
            call feeder.Execute("Move.Right",mvCenterDuration)
        Case 3
            call feeder.Execute("Move.BackwardRight",mvCenterDuration)
        Case 4
            call feeder.Execute("Move.Forward",mvCenterDuration)
        Case 5
            'no vibration
        Case 6
            call feeder.Execute("Move.Backward",mvCenterDuration)
        Case 7
            call feeder.Execute("Move.ForwardLeft",mvCenterDuration)
        Case 8
            call feeder.Execute("Move.Left",mvCenterDuration)
        Case 9
            call feeder.Execute("BackwardLeft", mvCenterDuration)
    End Select

    'Spreading the part by doing the flip

    call feeder.Execute("Move.Flip",300)
    'Disconnect
    Disconnect :
    cao.Controllers.Remove feeder.Index
    feeder = Nothing
    Exit Sub

    ErrorMgmt:
    ' Treat the errors here
    PrintMsg "Error..."
    Resume Disconnect

End Sub
```

# 5. Annex I. List of configuration & management commands

Warning! These are advance commands that could set the feeder in some incoherent settings if misused.

| # | **<Command>** | **<Value>** | **Description** |
|---|---|---|---|
| **1** | "*[!]*Bulk.Batch[<batchCode>].Start" | Duration [ms] [1] | Vibrate the bulk for the given duration [ms] with the batch parameters given by the <batchCode> argument. [a…z] |
| **2** | "*[!]*Platform.Batch[<batchCode>].Start" | Duration [ms] [1] | Vibrate the platform for the given duration [ms] with the batch parameters given by the <batchCode> argument. [a…z] |
| **3** | "Bulk.Batch[<batchCode>].Read" | - | Returning the parameters of a given bulk batch as a String.<br>The string is formatted as a "vector" with semicolon separating values.<br>The number of values depends on the feeder type. See the programming manual for more detail. (Command LB[A..Z]) |
| **4** | "Bulk.Batch[<batchCode>].Write" | - | Setting the parameter of a given bulk batch as a String.<br>The string argument must be formatted as a "vector" with semicolon separating values.<br>The number of values depends on the feeder type. See the programming manual for more detail. (Command SB[A..Z]) |
| **5** | "Platform.Batch[<batchCode>].Read" | - | Returning the parameters of a given platform batch as a String.<br>The string is formatted as a "vector" with semicolon separating values.<br>The number of values depends on the feeder type. See the programming manual for more detail. (Command LC[A..Z]) |
| **6** | "Platform.Batch[<batchCode>].Write" | - | Setting the parameter of a given platform batch as a String.<br>The string argument must be formatted as a "vector" with semicolon separating values.<br>The number of values depends on the feeder type. See the programming manual for more detail. (Command SC[A..Z]) |
| **7** | "Register[<RegNo>].Read" | - | Returning the value of the given register given by the <RegNo> argument.<br>The RegNo must be an even integer<br>See the "Programming manual" for more details |

| | | | |
|---|---|---|---|
| **8** | "Register[<RegNo>].Write" | Register value [Integer] | Writing the value of the register given by the argument <RegNo><br>Warning: RegNo must be an even integer. The provider is calculating the odd address (RegNo + 1) internally.<br>See the "Programming manual" for more details. |
| **10** | "GetLibVersion" | - | Returning the provider's software version as a string.<br>Following the format:<br>[libName=AsyucbeDILProv]-[LibVersion=<*version label*>]-[ReleaseDate=<*release date*>] |

*[!]* *C*ommand that accept the optional "!" prefix that makes an unblocking call (asynchronous)