



Helmut 4 - Nodes and Wildcards Guide

Software Documentation
Copyright © 2022 by MoovIT GmbH

1. Introduction	11
2. Overview	11
2.1. What is a node in Helmut4	11
2.1.1. Condition Nodes	11
2.1.1.1. Project Conditions:.....	12
2.1.1.2. User Conditions:.....	12
2.1.2. Action Nodes	12
2.1.2.1. Project Actions:.....	12
2.1.2.2. Job Actions:.....	13
2.1.3. Output Nodes	13
2.1.3.1. User Output:.....	13
2.2. What is a wildcard in Helmut4.....	14
3. Nodes and Workflow Examples.....	15
3.1. Condition Nodes.....	15
3.1.1. Project Conditions	15
3.1.1.1. Project Category Condition.....	15
3.1.1.2. Project Creator Condition.....	15
3.1.1.3. Project Extension Condition.....	15
3.1.1.4. Project Group Condition.....	16
3.1.1.5. Project Name Condition.....	16
3.1.1.6. Project Personal Condition.....	16
3.1.1.7. Project Template Condition.....	16
3.1.1.8. Project Version of Premiere Condition.....	16
3.1.2. User Conditions.....	18
3.1.2.1. User Displayname Condition.....	18
3.1.2.2. User Group Condition.....	18
3.1.2.3. User Role Condition.....	18
3.1.2.4. User Name Condition.....	18
3.1.3. File and Folder Conditions.....	19
3.1.3.1. File Content Condition.....	19
3.1.3.2. File Exists Condition.....	19
3.1.3.3. File Growing Condition.....	19
3.1.3.4. File Name Condition.....	20
3.1.3.5. File Size Condition.....	20
3.1.3.6. Folder Empty Condition.....	20
3.1.3.7. Folder Exists Condition.....	20
3.1.3.8. Folder Name Condition.....	20
3.1.4. OS Conditions	21

3.1.4.1. Operating System Condition	21
3.1.5. MediaInfo Conditions	21
3.1.5.1. Audio Channel Condition	21
3.1.5.2. Audio Streams Condition	21
3.1.5.3. Bitrate Codec Condition	21
3.1.5.4. FPS Condition	22
3.1.5.5. Media Codec Condition	22
3.1.5.6. Media Color Space Condition	22
3.1.5.7. Media Length Condition	22
3.1.5.8. Media Resolution Condition	22
3.1.6. MISC Conditions	23
3.1.6.1. Empty String Condition	23
3.1.6.2. IP Condition	23
3.1.6.3. Regex Match Condition	23
3.1.6.4. Wildcard Condition	23
3.1.6.5. Match Multiple Patterns Condition	23
3.2. Action Nodes	24
3.2.1. Project Action Nodes	24
3.2.1.1. Project Create Action	24
3.2.1.2. Project File Copy Action	24
3.2.1.3. Project Metadata Changer Action	24
3.2.1.4. Project File Download Action	25
3.2.1.5. Project File Upload Action	25
3.2.1.6. Project Metadata Remove Action	25
3.2.1.7. Project Set Category Action	25
3.2.1.8. Project Set Group Action	26
3.2.1.9. Project Set Name Action	26
3.2.1.10. Project Set Path Action	26
3.2.1.11. Project Set Tag Action	26
3.2.1.12. Project Set Template Action	26
3.2.1.13. Project as JSON Action	27
3.2.1.14. Project Delete Action	27
3.2.1.15. Project from JSON Action	27
3.2.1.16. Project Import Action	27
3.2.1.17. Project Import to External Helmut Action	27
3.2.1.18. Project Lock Action	28
3.2.1.19. Project Status Update Action	28
3.2.1.20. Project Update Modification Date Action	28
3.2.1.21. Project Duplicate Action	28
3.2.2. Job Action Nodes	29

3.2.2.1. Job Metadata Remove Action	29
3.2.2.2. Job Metadata Changer Action	29
3.2.2.3. Job Priority Action	29
3.2.2.4. Job QScan Action	29
3.2.2.5. Job Render AAF in Premiere Action	30
3.2.2.6. Job Render in AME Action.....	30
3.2.2.7. Job Render Proxy in AME Action	30
3.2.2.8. Job Render Premiere Action	30
3.2.2.9. Job Render with FFMEPG Action	31
3.2.2.10. Job Start Premiere Action	31
3.2.2.11. Job Status Update Action	31
3.2.2.12. Job as JSON Action	31
3.2.2.13. Job from JSON Action.....	32
3.2.2.14. Job Set Project ID Action	32
3.2.2.15. Job Create Job Action.....	32
3.2.2.16. Job Delete Action.....	32
3.2.2.17. Job Execute Extendscript in Premiere Action	32
3.2.2.18. Job File Copy Action	33
3.2.2.19. Job Folder Copy Action.....	33
3.2.2.20. Job RClone Copy Action.....	33
3.2.2.21. Job RClone Mkdir Action	33
3.2.2.22. Job RClone Move Action.....	33
3.2.2.23. Job RClone Purge Action.....	34
3.2.2.24. Job RClone Rmdir Action	34
3.2.2.25. Job Export FCP XML in Premiere	34
3.2.2.26. Job Create Split Jobs Action (BETA).....	34
3.2.2.27. Job Stitch Splits Action (BETA)	35
3.2.3. Premiere Action Nodes.....	36
3.2.3.1. Premiere Generate UUID Action	36
3.2.3.2. Premiere Alert Dialog Action.....	36
3.2.3.3. Premiere Confirm Dialog Action	36
3.2.3.4. Premiere Force Native Lock Action.....	36
3.2.3.5. Premiere Native Lock Action	36
3.2.3.6. Premiere OS Path Mapper Action	36
3.2.3.7. Premiere Open Choose Dialog Action	37
3.2.3.8. Premiere Path Settings Action	37
3.2.3.9. Premiere Prompt Dialog Action	37
3.2.3.10. Premiere Start Helmut Panel Action	37
3.2.3.11. Premiere Version Converter Action	37
3.2.4. Cosmo Action Nodes	38

3.2.4.1. Cosmo Add Asset To Project Action	38
3.2.4.2. Cosmo Add Info To Sequence Action	38
3.2.4.3. Cosmo Add Proxy To Project Action	38
3.2.4.4. Cosmo Change Asset Action.....	38
3.2.4.5. Cosmo Get Project Assets Action.....	39
3.2.4.6. Cosmo Change Project Asset Action	39
3.2.4.7. Cosmo Project File Index Action	39
3.2.4.8. Cosmo Get Project Sequences Action	40
3.2.4.9. Cosmo Asset Report Action.....	40
3.2.4.10. Cosmo Sequence Report Action	40
3.2.4.11. Cosmo Change Asset Metadata Action	40
3.2.4.12. Cosmo Delete Asset Metadata Action	40
3.2.4.13. Cosmo Get Asset Metadata Action	41
3.2.5. After Effects Action Nodes	42
3.2.5.1. After Effects OS Path Mapper Action	42
3.2.5.2. After Effects Alert Dialog Action	42
3.2.5.3. After Effects Open Choose Dialog Action	42
3.2.5.4. After Effects Prompt Dialog Action.....	42
3.2.5.5. After Effects Start Helmut Panel Action.....	42
3.2.5.6. Job Execute ExtendScript In After Effects Action	42
3.2.5.7. Job Start After Effects Action.....	43
3.2.6. Third Party Action Nodes.....	44
3.2.6.1. Editshare Set ACL Action	44
3.2.6.2. Editshare Delete ACL Action.....	44
3.2.6.3. EFS Create User Action	44
3.2.6.4. EFS Delete User Action	44
3.2.6.5. EFS Add Users To Group Action.....	44
3.2.6.6. EFS Remove User From Group Action.....	44
3.2.6.7. EFS Add Users To Media Space Action	44
3.2.6.8. EFS Remove Users From Media Space Action	45
3.2.6.9. EFS Create Group Action	45
3.2.6.10. EFS Remove Group Action.....	45
3.2.6.11. EFS Add Group To Media Space Action	45
3.2.6.12. EFS Remove Group From Media Space Action.....	45
3.2.6.13. EFS Create Media Space Action.....	45
3.2.6.14. EFS Delete Media Space Action.....	45
3.2.6.15. EFS Update Media Space Action.....	45
3.2.6.16. EFS Get Media Space DATA Action	45
3.2.6.17. EFS Get Free Space Of Media Space Action.....	46
3.2.6.18. Hue Bridge Group On Action	46

3.2.6.19. Hue Bridge Group Change Action	46
3.2.6.20. Hue Bridge Group Off Action	47
3.2.6.21. Hue Bridge Lamp On Action	47
3.2.6.22. Hue Bridge Lamp Change Action.....	47
3.2.6.23. Hue Bridge Lamp Off Action	47
3.2.6.24. CatDV Add Asset to Catalog Action	47
3.2.6.25. CatDV Create Catalog Path Action.....	47
3.2.6.26. CatDV Delete Catalog Path Action	48
3.2.6.27. Flow Add Asset To Project Action	48
3.2.6.28. Flow Create Path Action.....	48
3.2.6.29. Flow Delete Path Action.....	48
3.2.6.30. Flow Toggle Private Project Action.....	48
3.2.6.31. SwatIO Job Upload Action	48
3.2.6.32. Medialoopster Add Asset To Project Action	49
3.2.6.33. Medialoopster Create Project Action.....	49
3.2.6.34. Medialoopster Delete Asset From Project Action.....	49
3.2.6.35. Medialoopster Delete Project Action	49
3.2.6.36. Medialoopster Update Project Delete Date Action	49
3.2.6.37. Mount EFS Volume (MacOS) Action	49
3.2.6.38. Mount EFS Volume (Windows) Action	49
3.2.6.39. Stratus Create Asset Action.....	50
3.2.6.40. Stratus Create Project Action.....	50
3.2.6.41. Stratus Generate Unique ID Action	50
3.2.6.42. Stratus Patch Asset Action.....	50
3.2.6.43. Stratus Project Indexed Action	50
3.2.6.44. Stratus Project Status Action	50
3.2.6.45. Stratus Transfer Asset Action	50
3.2.6.46. Stratus Trigger Momentum Action.....	50
3.2.6.47. Stratus Add Asset to Folder Action	50
3.2.6.48. Stratus Get All Folders Action.....	50
3.2.6.49. Stratus Create Folder Action	51
3.2.6.50. Stratus Update Project Path Action.....	51
3.2.6.51. Stratus Asset Add Folder Membership Action	51
3.2.6.52. Aurora Submit Job Action	51
3.2.6.53. RevApp Share Asset By Email Action	51
3.2.6.54. RevApp Share Asset Public Link Action.....	51
3.2.6.55. RevApp Share Asset with Internal Group Action	51
3.2.6.56. RevApp Upload Asset To Namespace Action	51
3.2.6.57. RevApp Delete Asset From Namespace Action.....	51
3.2.6.58. Hiscale Jobs Start Process Action	51

3.2.6.59. VPMS Check-in Action	52
3.2.6.60. VPMS Metadata Update Action.....	52
3.2.6.61. Elements Add Permissions To Workspace Action	52
3.2.6.62. Elements Add Users To Groups Action.....	52
3.2.6.63. Elements Compare To Helmut User Action.....	52
3.2.6.64. Elements Copy Files Action	52
3.2.6.65. Elements Create API token Action	53
3.2.6.66. Elements Create Groups Action	53
3.2.6.67. Elements Create Directories Action	53
3.2.6.68. Elements Create Media Root Action.....	53
3.2.6.69. Elements Create Production Action	53
3.2.6.70. Elements Create Share Action.....	54
3.2.6.71. Elements Create Users Action	54
3.2.6.72. Elements Create Workspace Action.....	54
3.2.6.73. Elements Delete Files Action.....	54
3.2.6.74. Elements Delete Files From Media Library Action	54
3.2.6.75. Elements Delete Groups Action	55
3.2.6.76. Elements Delete Directories Action	55
3.2.6.77. Elements Delete Media Root Action.....	55
3.2.6.78. Elements Delete Production Action	55
3.2.6.79. Elements Delete Share Action.....	55
3.2.6.80. Elements Delete Users Action	55
3.2.6.81. Elements Delete Workspaces Action.....	55
3.2.6.82. Elements Get Client Key Action	55
3.2.6.83. Elements Get Free Space Of Workspace Action.....	56
3.2.6.84. Elements HTTP Request Action	56
3.2.6.85. Elements Mount Workspaces On Client Action.....	56
3.2.6.86. Elements Move Files Action	56
3.2.6.87. Elements Remove Permissions From Workspace Action	56
3.2.6.88. Elements Remove Users From Groups Action	56
3.2.6.89. Elements Start Job Action	56
3.2.6.90. Elements Unmount Workspaces From Client Action	57
3.2.6.91. Elements Update Workspace Quota Action.....	57
3.2.6.92. Elements Upload File Into Media Library Action	57
3.2.7. User.....	58
3.2.7.1. Helmut Add Users To Group Action.....	58
3.2.7.2. Helmut Remove Users From Group Action.....	58
3.2.7.3. Helmut Input Dialog Action.....	58
3.2.7.4. Helmut Confirm Dialog Action.....	58
3.2.8. File and Folder Actions	58

3.2.8.1. File Copy Action	58
3.2.8.2. File Create Action	58
3.2.8.3. File Delete Action	58
3.2.8.4. File Increment Name Action	58
3.2.8.5. File Move Action	59
3.2.8.6. File Open Action	59
3.2.8.7. File Rename Action	59
3.2.8.8. File Replace Content Action	59
3.2.8.9. Folder Content Delete Action	59
3.2.8.10. Folder Copy Action	59
3.2.8.11. Folder Create Action	59
3.2.8.12. Folder Delete Action	59
3.2.8.13. Folder Increment Name Action	60
3.2.8.14. Folder Move Action	60
3.2.8.15. Folder Rename Action	60
3.2.8.16. XSquare File Check In Action	60
3.2.8.17. 7-Zip Archive Action	60
3.2.8.18. 7-Zip Extract Action	60
3.2.8.19. 7-Zip CLI Action	60
3.2.8.20. Get Files From Folder Action	60
3.2.8.21. Get Folder From Folder Action	60
3.2.9. OS Action Nodes	61
3.2.9.1. Commandline Execute Action	61
3.2.9.2. Unmount A Share Action	61
3.2.10. MISC Action Nodes	61
3.2.10.1. Execute Javascript Action	61
3.2.10.2. Fail Action	61
3.2.10.3. HTTP Request Action	61
3.2.10.4. JSON Extract Action	61
3.2.10.5. Metadata Auto Mapper Action	63
3.2.10.6. Regex Apply Action	63
3.2.10.7. Sleep Action	63
3.2.10.8. Split Stream Action	63
3.2.10.9. Stream Execute Generic Stream Action	63
3.2.10.10. Stream Set Temporary Variable Action	63
3.2.10.11. Stream Set Store Variable Action	64
3.2.10.12. Stream Get Store Variable Action	64
3.2.10.13. Stream Delete Store Variable Action	64
3.2.10.14. Success Action	64
3.2.10.15. XPath Action	64

3.2.10.16.XML Generator Action.....	64
3.3. Output Nodes.....	65
3.3.1. File and Folder Output.....	65
3.3.1.1. Write File Output.....	65
3.3.2. OS Output.....	65
3.3.2.1. MacOS System Notification Output.....	65
3.3.3. MISC.....	65
3.3.3.1. Send Email Output.....	65
3.3.3.2. Telegram Output.....	65
3.3.3.3. Send Message To Teams Channel.....	66
3.3.4. User.....	70
3.3.4.1. Send Message to User	70
3.3.4.2. Send Notification to User.....	70
4. Wildcards	71
4.1. General Wildcards.....	71
4.1.1. Job Breadcrumb Wildcard.....	71
4.1.2. Job Destination Wildcard.....	71
4.1.3. Job Proxy Wildcard	71
4.2. Functional Wildcards	71
4.2.1. Convert Date To Timestamp ? Wildcard.....	71
4.2.2. Convert Timestamp To Date ? Wildcard.....	72
4.2.3. Convert Timestamp To Datetime ? Wildcard.....	72
4.2.4. Date Day ? Wildcard	72
4.2.5. Date Month ? Wildcard.....	72
4.2.6. Date Month Textual ? Wildcard	72
4.2.7. Date Month Textual Short ? Wildcard	72
4.2.8. Date Year ? Wildcard.....	73
4.2.9. Date Shortyear ? Wildcard	73
4.2.10. File Content ? Wildcard.....	73
4.2.11. File Exists ? Wildcard.....	73
4.2.12. File MD5 ? Wildcard.....	73
4.2.13. File Modified ? Wildcard	74
4.2.14. File Size ? Wildcard.....	74
4.2.15. Folder Content ? Wildcard	74
4.2.16. Folder Exists ? Wildcard	74
4.2.17. Folder Modified ? Wildcard.....	75
4.2.18. Folder Size ? Wildcard	75
4.2.19. Helmut Variable ? Wildcard	75
4.2.20. Job metadata ? Wildcard	75

4.2.1. Job custom ? Wildcard	76
4.2.1. Project custom ? Wildcard	76
4.2.2. Local Environment ? Wildcard.....	76
4.2.3. Local Profile ? Wildcard	76
4.2.4. Local Registry ? Wildcard	77
4.2.5. Path Basename ? Wildcard	77
4.2.6. Path Basename ? Wildcard	77
4.2.7. Path Map To Unix ? Wildcard	77
4.2.8. Path Map To Win ? Wildcard	78
4.2.9. Path Map Auto ? Wildcard.....	78
4.2.10. Path Map To JSON ? Wildcard	78
4.2.11. Path Name ? Wildcard	78
4.2.12. Path Parent ? Wildcard.....	78
4.2.13. Path Split ? Wildcard.....	79
4.2.14. Project Metadata? Wildcard	79
4.2.15. Stream Variable ? Wildcard	80
4.2.16. String Split ? Wildcard.....	80
4.2.17. String length ? Wildcard	81
4.2.18. String Case To Camel ? Wildcard.....	81
4.2.19. String Case To Kebab ? Wildcard.....	81
4.2.20. String Case To Lower ? Wildcard	81
4.2.21. String Case To Pascal ? Wildcard.....	81
4.2.22. String Case To Snake ? Wildcard	81
4.2.23. String Case To Upper ? Wildcard	82
4.2.24. String Node Result ? Wildcard	82
4.2.25. String Node Result ? Wildcard	82

1. Introduction

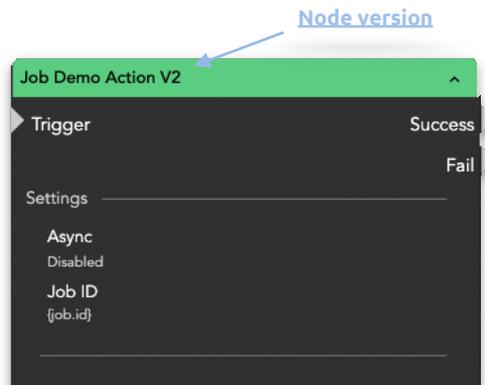
In order to understand the Stream Designer and all the abundance and functionality that Helmut4 offers, and to use it correctly, it is necessary to understand how nodes work and can be logically linked. This guide does not claim to provide a complete documentation of all possible combinations, but rather represents a guide for in-depth knowledge transfer.

2. Overview

2.1. What is a node in Helmut4

There are 3 types of nodes in Helmut 4. Condition nodes, action nodes and output nodes. Each node can be used as often as required within a stream, but the sequence of nodes should represent a logical connection. For example, after you have created a project (create project action), it makes no sense to try to export this project with AME (job render in AME action) because AME only understands files or sequences from a project, and not the project itself.

To get an overview about all used nodes within the stream designer it is helpful to trace the current version of certain nodes. Therefore the suffix with the version number (e.g. V2) is part of the node's name. The user can always be aware of the current version and can check possible existing updates of a node.



2.1.1. Condition Nodes

A condition node enables the further course of a stream (workflow) to be linked to one or more conditions. There are different types of conditions that apply depending on which trigger triggers the stream (workflow). For example:

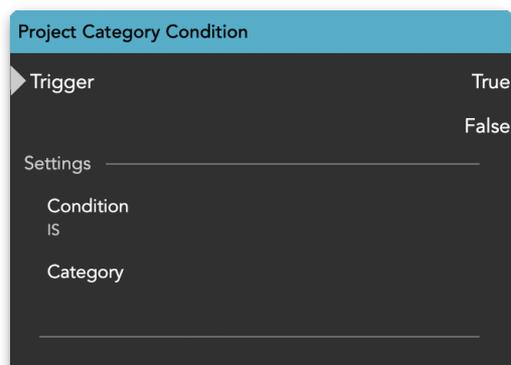


Figure 2.1. Condition Node
Helmut Streams - V. 4.3.0 - 20.07.2022

2.1.1.1. Project Conditions:

A project condition always refers to the information that is / was assigned to a project. This means, for example, that it is possible to check whether a project belongs to a certain group or who created the project. If a decision is to be made in the stream on the basis of this information, it is queried via a condition action and checked for true or false. Example:

If a project is created via the "add project" dialog in HelmutFX and the selected group in the dialog (this field must be filled out) is called "Sport", it is possible to query this decision made by the user in the stream and to set a condition for the further process of the workflow. For example, the fact that the selected group is "Sport" could mean that different Action Nodes are executed for this group than for all other groups that were created in the system.

2.1.1.2. User Conditions:

A user condition always refers to the information available about a user executing a stream. This means, for example, that the name of the user or his group membership can be made a condition for the further course of the stream. Example:

If there are streams that are configured to always be executed via the local client (autoimport), a different decision can be made in the stream depending on the user. For example, if the user is called "Lee", the user name condition can be used to query whether Lee is the one executing the stream or not. If this is the case, the further course of the stream can change due to the fact that the condition is true.

2.1.2. Action Nodes

An action node is used to carry out an action either via the client (local client on the workstation) or via the server. The possible actions as well as the possible conditions are summarized in logical groups and not every action makes sense for every trigger. The prerequisites for the logical application of an action are many and vary from case to case. An Action can be executed synchronously or asynchronously, depending on the amount of time the action takes. Here are two examples:



Figure 2.2. Action Node

2.1.2.1. Project Actions:

A project action always refers to a project object, be it that it is being created, has already been created, or is to be changed. It is therefore possible to use a project action whenever there is project-related information in the stream, i.e. the trigger is related to a project. Example:

If a project is created, all project-related information entered in the "add project" dialog is available in the stream and can be used in an action node. If the project object is to be written to a certain location, for example, the "project file copy action" can be used and the information that was defined in the input

dialog can be used in this. It would also be conceivable to create the project in a third party application and to use the same or a different name that was entered for the project.

2.1.2.2. Job Actions:

A job action always relates to a job object, i.e. to a job executed via IO, which in turn always relates to an asset (file or sequence). Since a job can also be triggered from within a project, there is the case that project-related information (name of the project, metadata, etc.) is also available in a job.

A job is always shown in the IO dashboard. Metadata as well as the progress and a message are displayed. If, for example, an import of a file is initiated, the "job file copy action" can be used to copy this file to a defined location on the storage system. The progress is shown in the dashboard. If this file is to be recoded after it has reached the destination, the "job render in name action" can be used to do this. Again, the progress is displayed in the dashboard.

2.1.3. Output Nodes

Output nodes are there to carry the information that is within a stream at any point to the outside, for example to inform a user or administrator, or simply to write the information in the stream to a file. Information present in the stream can be any static or any variable information. Example:

If you want to inform a certain person about the progress of the export during an export, it is possible to write an email to the person concerned. This can be done after each node. It is also conceivable to write all information about each individual processed asset to a file during an import in order to evaluate this information in a later step.

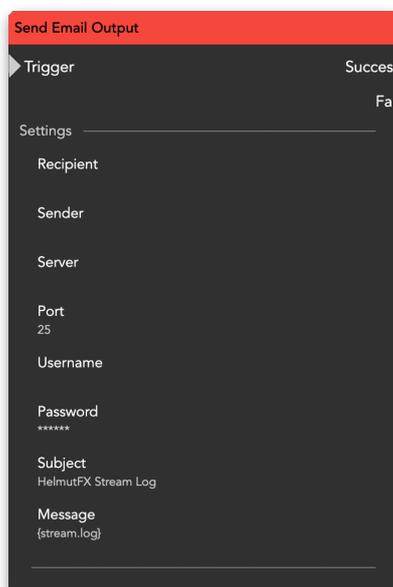


Figure 2.3. Output Node

2.1.3.1. User Output:

A user output can be used to inform a defined user via message or notification. This can be done after each node in the stream. For example, if you want to inform the user whether an export has been successfully completed, this is possible via a "send message to user" node at the end of the stream.

2.2. What is a wildcard in Helmut4

Wildcards are used to dynamically change the behaviour of nodes within the stream. So that you don't have to use a new node for each condition, you can also choose a placeholder for the condition so that you only have to use a single node. Example:

If you want to control the further course of the stream in a stream that was triggered by the "create project" trigger using a condition that is true for a specific user, for example, you can use the placeholder {user.name}, and as source within the "wildcard condition" node. As a result, this placeholder (the source) is always replaced with the username that triggered the stream and can thus be compared against the desired value.

Another example is the predefined paths in Helmut4 -> Preferences. The path for projects is always used when the placeholder {helmut.projects} is used within a node. This makes configurations simpler and more flexible. Figure 2.4 shows how this works. The field for the value to be checked is filled with a wildcard, which extracts its value from a variable defined in the Preferences. The variable itself is named "AEPX" and has the value "After Effects". Wherever you want to write or use "After Effects" within the stream, it is possible to use the wildcard {helmut.variable.?.} And replace the question mark with the key "AEPX".

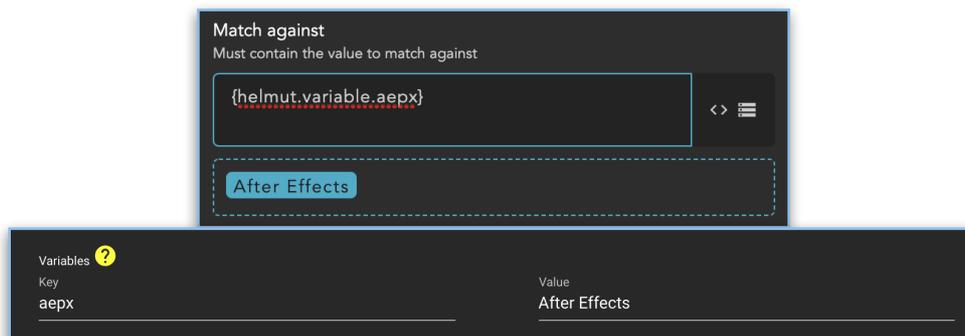


Figure 2.4. wildcard replacement

3. Nodes and Workflow Examples

3.1. Condition Nodes

3.1.1. Project Conditions

3.1.1.1. Project Category Condition

With the help of the Project Category Condition it is possible to use the category of the project for which the stream was triggered as a condition for further actions. For example, if you want to change the behaviour of the stream triggered by the "create project" trigger for a certain group, you would enter the name of the group for which the behaviour is to change in this node. For example, if the node is to be successful when it comes to a certain group, one would use "is" as the condition. If the node is to fail when it is a certain group, one would use "is_not". This makes it possible to decide precisely whether the behaviour should apply to a group or not.

The Project Category Condition can always be queried if the object for which the stream in which this node is used was triggered is a project.

3.1.1.2. Project Creator Condition

With the help of the Project Creator Condition, it is possible to use the creator of the project for which the stream was triggered as a condition for further actions. For example, if you want to change the behaviour of the stream triggered by the "duplicate project" trigger for a specific creator, you would enter the name of the creator for whom the behaviour is to be changed in this node. For example, if the node is to be successful if it is a particular creator, one would use "is" as the condition. If the node is to fail if it is a particular creator, "is_not" would be used. This makes it possible to decide precisely whether the behaviour should apply to a creator or not to a creator.

The Project Creator Condition can always be queried if the object for which the stream in which this node is used was triggered is a project.

3.1.1.3. Project Extension Condition

With the help of the Project Extension Condition it is possible to use the project extension of the project for which the stream was triggered as a condition for further actions. For example, if you want to make the behaviour of the stream triggered by the "duplicate project" trigger dependent on whether it is a specific project type, you would enter the name of the extension for which the behaviour should change in this node. Here the way can be defined. For example, if the node is to be successful if it is a certain project type, one would use "is" as a condition. If the node is to fail if it is a certain project type, it would be you can use "is_not". This makes it possible to decide precisely whether the behaviour should apply to a project type or not.

The project extension condition can always be queried if the object for which the stream in which this node is used was triggered is a project.

3.1.1.4. Project Group Condition

With the help of the Project Group Condition it is possible to use the group of the project for which the stream was triggered as a condition for further actions. For example, if you want to make the behaviour of the stream triggered by the "duplicate project" trigger dependent on whether it is a specific group, you would enter the name of the group in this node for which the behaviour is to change. Here the way can be defined. For example, if the node is to be successful if it is a specific group, one would use "is" as the condition. If the node is to fail if it is a specific group, it would be You can use "is_not" This makes it possible to decide precisely whether the behaviour should apply to a group or not.

The project group Condition can always be queried if the object for which the stream in which this node is used was triggered is a project.

3.1.1.5. Project Name Condition

With the help of the Project Name Condition it is possible to use the name or parts of the name of the project for which the stream was triggered as a condition for further actions. For example, if you want to make the behaviour of the stream triggered by the "edit project" trigger dependent on whether it is a specific project, you would enter the name, or parts of the name, of the project for which the behaviour should change. Here the way can be defined. For example, if the node is to be successful when it comes to a certain project, one would use "is" as the condition. If the node should fail if it is a specific project, one would use "is_not". This makes it possible to decide precisely whether the behaviour should apply to a project or not.

The project name condition can always be queried if the object for which the stream in which this node is used was triggered is a project.

3.1.1.6. Project Personal Condition

With the help of the Project Personal Condition it is possible to use the fact that a project is a private project as a condition for further actions. For example, if you want to make the behaviour of the stream triggered by the "delete project" trigger dependent on the fact that it is a private project, you would use "is" as the condition. If it should not depend on it, you would Use "is_not".

The project personal condition can always be queried if the object for which the stream in which this node is used was triggered is a project.

3.1.1.7. Project Template Condition

With the help of the Project Template Condition it is possible to use the template name or parts of the name of the project for which the stream was triggered as a condition for further actions. For example, if you want to make the behaviour of the stream triggered by the "duplicate project" trigger dependent on whether it is a specific template, you would enter the name, or parts of the name, of the template for which the behaviour should change. Here the way can be defined. For example, if the node is to succeed if it is a specific template, one would use "is" as the condition. If the node should fail if it is a If you are dealing with a certain template, you would use "is_not". This makes it possible to decide precisely whether the behaviour should apply to a template or not.

The project template condition can always be queried if the object for which the stream in which this node is used was triggered is a project.

3.1.1.8. Project Version of Premiere Condition

With the help of the Project Version of Premiere Condition it is possible to check the version of a certain Premiere project. The project version can be compared by providing the related „Project Path“ and a desired condition (e.g. IS_OLDER). The node result will be the corresponding version number.

Matching of Premiere version to version number:

V2022 (40), v2021 (39), v2020 (38), v2019_1 (37), v2019 (36), v2018_1 (35), v2018 (34), v2017_1 (33),
v2017 (32), v2015_4 (31), v2015_2 (30)

3.1.2. User Conditions

3.1.2.1. User Displayname Condition

With the help of the User Displayname Condition it is possible to use all or part of the display name of the user who is executing the stream as a condition for further actions. For example, if you want to make the behaviour of the stream dependent on whether the user has a certain display name, you would enter the display name or parts of the display name for which the behaviour should change in this node. Here the way can be defined. For example, if the node is to be successful when it comes to a specific user, "is" would be used as the condition. "Is_not" would be used if the node should fail for a specific user. This makes it possible to decide precisely whether the behaviour should apply to a user or not.

The user display name condition can be queried in every stream.

3.1.2.2. User Group Condition

With the help of the User Group Condition it is possible to use one or more groups to which the user executing the stream belongs, in whole or in part, as a condition for further actions. For example, if you want to make the behaviour of the stream dependent on whether the user belongs to a certain group, enter the group name or parts of the group name for which the behaviour in the stream should change. The way in which this should be done can be defined. For example, if the node is to be successful for a particular group, "is" is used as the condition. "Is_not" is used if the node should fail for a specific group. In this way it can be precisely decided whether the behaviour should apply to a group or not.

The user group condition can be queried in each stream.

3.1.2.3. User Role Condition

With the help of the User Role Condition it is possible to use the role of the user executing the stream as a condition for further actions. For example, if you want to make the behaviour of the stream dependent on whether the user is an administrator, select "Admin" from the dropdown menu. The way in which this should be done can be defined. For example, if the node should be successful if the user is an admin, "is" is used as a condition. "Is_not" is used if the node should fail in case the user is an admin. In this way it can be precisely decided whether the behaviour should apply to a group or not.

The user role condition can be queried in each stream.

3.1.2.4. User Name Condition

With the help of the Username Condition it is possible to use the name of the user executing the stream as a condition for further actions. If the node is to be successful for a particular user, "is" is used as the condition. "Is_not" is used when the node should fail for a specific user. In this way, it can be precisely decided whether the behaviour should apply to a user or not.

The user name condition can be queried in every stream.

3.1.3. File and Folder Conditions

3.1.3.1. File Content Condition

With the help of the File Content Condition it is possible to use the whole or parts of the content of a file as a condition for further actions. The path and the extension of the file as well as the part of the content that is defined as a condition are specified. This makes it possible to combine one workflow with another by, for example, writing information into a text file in one workflow, which can be read out again in another workflow and made a condition.

The file condition can be queried in every stream, as it is not linked to the project object or the user object.

3.1.3.2. File Exists Condition

With the help of the File Exists Condition it is possible to use the presence or absence of a file as a condition for further actions. The path and the extension of the file are specified here.

The File Exist Condition can be queried in every stream because it is not linked to the project object or the user object.

3.1.3.3. File Growing Condition

With the help of the File Growing Condition it is possible to use the fact whether a file is still in the write state or not as a condition for further actions. The path and the extension of the file are specified here. You enter the interval within which the node checks the file and the number of checks itself. For example, if 3 seconds are specified as the interval, 5 checks are defined and the condition is set to "is_growing", the following happens:

The node checks the file depending on the set condition. If it is determined that the file is still growing, it is checked again after 3 seconds. If the file still grows after the second check, the node has failed.

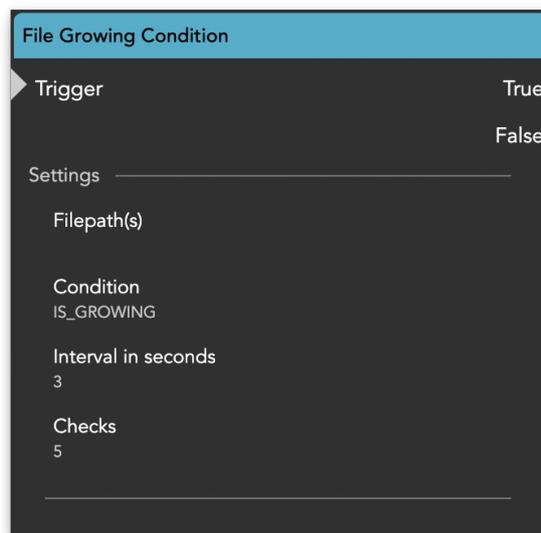


Figure 3.1.1. wildcard replacement

The node will check the file size in the defined interval, if the file size does not change for the amount of set Checks, the file is set as *not growing*" If the file size changes between the checks, it is set as *growing*. The node will not check infinitely, it is just meant to define the current state of a given file

The File Growing Condition can theoretically be queried in every stream, as it is not linked to the project object or the user object, but it usually makes sense if there is a job object in the stream. The job object is available if it is a HelmutIO stream.

3.1.3.4. File Name Condition

With the help of the File Name Condition it is possible to use the name or part of the name of one or more files as a condition for further actions. The path and the extension of the file (s) are specified here. The file whose name serves as the basis can in principle be any file that the client executing the stream can access.

The File Name Condition can theoretically be queried in every stream, as it is not linked to the project object or the user object, but it mostly makes sense if there is a job object in the stream. The job object is available if it is a HelmutIO stream.

3.1.3.5. File Size Condition

With the help of the File Size Condition it is possible to make the size of one or more files a condition for further actions. For example, it is conceivable to process all files that exceed a certain size differently than the files whose size is below the defined value.

The File Size Condition can theoretically be queried in every stream, as it is not linked to the project object or the user object, but it mostly makes sense if there is a job object in the stream. The job object is available if it is a HelmutIO stream.

3.1.3.6. Folder Empty Condition

With the help of the Folder Empty Condition it is possible to make the contents of one or more folders a condition for further actions. You can define whether you want to check whether one or more folders are empty or already filled.

The Folder Empty Condition can be queried in every stream.

3.1.3.7. Folder Exists Condition

With the help of the Folder Exists Condition it is possible to make further actions in the stream dependent on whether a folder exists or not. You can define whether you want to check whether one or more folders exist or not.

The Folder Exists Condition can be queried in every stream.

3.1.3.8. Folder Name Condition

With the help of the Folder Name Condition it is possible to use the name or parts of the name of one or more folders as a condition for further actions. The path is specified here. The folders whose name serves as the basis can in principle be any folder that the client executing the stream can access.

The Folder Name Condition can be queried in every stream.

3.1.4. OS Conditions

3.1.4.1. Operating System Condition

With the help of the Operating System Condition it is possible to use the operating system of the client that is executing the stream as a condition for further actions. Therefore it is mandatory to select the client as host in the event trigger. Here it is possible to check whether the operating system is Windows, Unix, Linux or MAC. This can be helpful e.g. for the right decision of the used AME version for rendering. Also it can be checked if the defined job profile can potentially be executed by different render nodes.

The Operating System Condition can be queried in every stream.

3.1.5. MediaInfo Conditions

3.1.5.1. Audio Channel Condition

With the help of the audio channel condition it is possible to use the number of audio channels in a video file as a condition for further actions. Helmut 4 uses Media Info for this, which is automatically installed when the client is installed. It is possible to define the number of channels that are or should be available. This condition is helpful if e.g. a proxy with the appropriate number of audio tracks is to be generated for each video file. (Mandatory requirement for Premiere Pro Proxy workflows).

The Audio Channel Condition can theoretically be queried in every stream, as it is not linked to the project object or the user object, but it mostly makes sense if there is a job object in the stream. The job object is available if it is a HelmutIO stream.

3.1.5.2. Audio Streams Condition

With the help of the audio streams condition it is possible to use the number of audio streams in a video file as a condition for further actions. Helmut 4 uses Media Info for this, which is automatically installed when the client is installed. It is possible to define the number of streams that are or should be available. This condition is helpful if e.g. a proxy with the appropriate number of audio tracks is to be generated for each video file. (Mandatory requirement for Premiere Pro Proxy workflows). Due to the fact that a distinction is made between audio channels and audio streams within an .mxf file, this condition exists alongside the audio channel condition.

The Audio Stream Condition can theoretically be queried in every stream, as it is not linked to the project object or the user object, but it mostly makes sense if there is a job object in the stream. The job object is available if it is a HelmutIO stream.

3.1.5.3. Bitrate Codec Condition

With the help of the bit rate codec condition it is possible to use the bit rate of a video file as a condition for further actions. Helmut 4 uses Media Info for this, which is automatically installed when the client is installed. It is possible to define the bit rate in mbit as an integer. This condition is helpful if e.g. For each video file that exceeds a certain bit rate, a high res or proxy should be generated so as not to overuse the storage bandwidth.

The Bitrate Codec Condition can theoretically be queried in every stream, as it is not linked to the project object or the user object, but it mostly makes sense if there is a job object in the stream. The job object is available if it is a HelmutIO stream.

3.1.5.4. FPS Condition

With the help of the FPS condition it is possible to use the number of frames per second of a video file as a condition for further actions. Helmut 4 uses Media Info for this, which is automatically installed when the client is installed. It is possible to define the FPS as an integer. This condition is helpful if e.g. a new file should be created for each video file that has an FPS that differs from the house standard. For example, for videos from cell phones.

The FPS Condition can theoretically be queried in every stream, as it is not linked to the project object or the user object, but it mostly makes sense if there is a job object in the stream. The job object is available if it is a HelmutIO stream.

3.1.5.5. Media Codec Condition

With the help of the Media Codec Condition it is possible to use the codec of a video file as a condition for further actions. Helmut 4 uses Media Info for this, which is automatically installed when the client is installed. It is possible to define the codec in the form of a string. The exact designation and spelling of the possible codecs is defined by Media Info. This condition is helpful, for example, if every video file that deviates from the house standard is to be recoded.

The Media Codec Condition can theoretically be queried in every stream, as it is not linked to the project object or the user object, but it mostly makes sense if there is a job object in the stream. The job object is available if it is a HelmutIO stream.

3.1.5.6. Media Color Space Condition

With the help of the Media Color Space Condition it is possible to use the color space of a video file as a condition for further actions. Helmut 4 uses Media Info for this, which is automatically installed when the client is installed. It is possible to define the color space in the form of a string. The exact designation and spelling required for this is defined by Media Info.

The Media Color Space Condition can theoretically be queried in every stream, as it is not linked to the project object or the user object, but it mostly makes sense if there is a job object in the stream. The job object is available if it is a HelmutIO stream.

3.1.5.7. Media Length Condition

With the help of the Media Length Condition it is possible to use the length of a video file as a condition for further actions. Helmut 4 uses Media Info for this, which is automatically installed when the client is installed. It is possible to define the length in seconds in the form of an integer. A possible workflow would be, for example, for each file that exceeds a defined length, to outsource the associated copying or rendering process to a separate machine.

The Media Length Condition can theoretically be queried in every stream, as it is not linked to the project object or the user object, but it mostly makes sense if there is a job object in the stream. The job object is available if it is a HelmutIO stream.

3.1.5.8. Media Resolution Condition

With the help of the Media Resolution Condition it is possible to use the resolution of a file as a condition for further actions. Helmut 4 uses Media Info for this, which is automatically installed when the client is installed. It is possible to define the resolution both in height and in width in the form of an integer. A possible workflow would be to forbid the import into the system for every file whose resolution is not 16:9. The Media Length Condition can theoretically be queried in every stream, as it is not linked to the project object or the user object, but it mostly makes sense if there is a job object in the stream. The job object is available if it is a HelmutIO stream.

3.1.6. MISC Conditions

3.1.6.1. Empty String Condition

With the help of the Empty String Condition it is possible to check if a source like the value from a Helmut variable, project or job metadata field is empty or not. In addition to this validation, it is also possible to handle null source values as if it is an empty string. This would make it possible, for example, to find if a given Helmut metadata field is empty or not.

3.1.6.2. IP Condition

With the help of the IP Condition it is possible to use the IP addresses available on the client's network adapter as a condition for further actions. This would make it possible, for example, to find out whether the client is registered in the "home network" or remotely and to change the workflows accordingly.

The IP Condition can be queried in every stream.

3.1.6.3. Regex Match Condition

With the help of the Regex Match Condition it is possible to match one or more source files against a REGEX. This makes it easy to filter out files in a pre-stream, for example, that contain characters that are not allowed.

3.1.6.4. Wildcard Condition

With the help of the Wildcard Condition it is possible to replace any other condition, respectively to match any source against any value. This value can be fixed or defined via a wildcard.

For example, it would be conceivable to use this to replace the user condition by simply entering the user name and matching it with {user.name}. Since this node exists, this is of course pointless. A more sensible use case would be to match the {job.source} in a stream against a predefined extension, for example .MXF.

Be creative.

3.1.6.5. Match Multiple Patterns Condition

With the help of the Match Multiple Patterns Condition it is possible to match the input against one or multiple Regular Expressions (not case sensitive). Within the node settings there is the possibility to select a certain condition and decide if the input must match the condition for only „ONE“ or for „ALL“ patterns. The „Input“ will be provided by the user and contains the string to match against the pattern(s). On the other the „Pattern(s)“ should be provided and allows regular expressions. More pattern fields can be added by clicking the Plus Button on the right hand side.

3.2. Action Nodes

3.2.1. Project Action Nodes

3.2.1.1. Project Create Action

With the help of the Project Create Action it is possible to create a project without using the "Add Project" dialog in Helmut FX. The node itself triggers the streams whose trigger is "create_project" and transfers the parameters required for this. These are:

- name
- group
- category
- template
- project type

It is imperative that the template which is defined is physically present in the group that has been selected. This makes it possible, for example, in a workflow based on an existing project object to generate a copy in which the parameters defined in the node from the project object are used in the form of wildcards. Another conceivable variant would be to create a project in Helmut in which an XML is read from a third party system via a watch folder workflow.

The Project Create Action can be queried in every stream in which a project object is present.

3.2.1.2. Project File Copy Action

With the help of the Project File Copy Action it is possible while a project is being created in the database to create the physical file on the storage from the template.

Example: If all parameters are defined in the "Add project" dialog that are necessary to create a project, this only means that an object will be created in the database. (Metadata only). If a workflow is also executed on the associated trigger "create_project", the physical equivalent of the database object on the storage is only defined via the workflow.

It is therefore only possible to open a project when this node has generated the physical file from the template.

The File Copy Action can be queried in every stream in which a project object is present.

3.2.1.3. Project Metadata Changer Action

With the help of the Project Metadata Changer Action it is possible to change an existing metadata entry on the project object within a workflow. This can be used, for example, to change the status of a project when certain work steps have been carried out.

- Project was opened
- Projects was closed
- Sequence has been exported
- Etc.

Due to the completely free definition of metadata, a status could also be freely defined.

The Project Metadata Changer Action can be queried in every stream in which a project object is present.

3.2.1.4. Project File Download Action

With the help of the Project File Download Action it is possible to download a project file to the local workstation. This makes it possible to open a project even if there is no connection to the drive on which the project file is saved. For example, the path: / Users / Public / Helmut is recommended under MAC, as this is available on every system.

Within the Open project stream, the project file would be written to this location and opened from there.

3.2.1.5. Project File Upload Action

With the help of the Project File Upload Action it is possible to upload a project file that has been downloaded to the local workstation and edited there again to the server. It makes sense to use this node in a stream that is linked to the unlock_project trigger.

The file is automatically linked on the server using the UUID.

3.2.1.6. Project Metadata Remove Action

With the help of the Project Metadata Remove Action it is possible to remove a metadata entry that has already been set on a project. This only applies to custom metadata. Custom project metadata is displayed in FX, CO and HK when you click in the project table between the displayed values. See Figure 3.1.

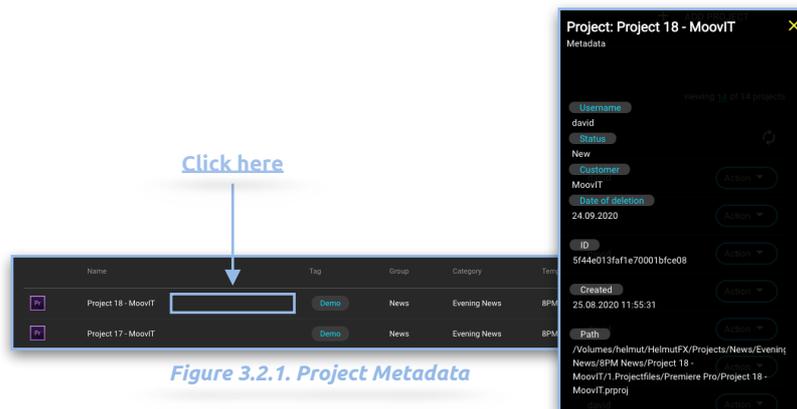


Figure 3.2.1. Project Metadata

The Project Metadata Remove Action can be queried in every stream in which a project object is present.

3.2.1.7. Project Set Category Action

With the help of the Project Set Category Action it is possible to (re) set the category of a project.

This can be used, for example, to sort projects differently in the view in FX. Once defined, the category is simply a metadata entry that can be called up using a corresponding wildcard in the stream designer. Due to this fact, workflows can be directly dependent on the project category and therefore it might make sense to want to change the category of a project.

The Project Set Category Action can be queried in every stream in which a project object is present.

3.2.1.8. Project Set Group Action

With the help of the Project Set Group Action it is possible to (re) set the group of a project.

This can be used, for example, to sort projects differently in the view in FX, as the group membership of a project determines who is allowed to see a project and who is not.

If you want to move a project back and forth between groups, it would be possible to change the access via this node.

The Project Set Group Action can be queried in every stream in which a project object is present.

3.2.1.9. Project Set Name Action

With the help of the Project Set Name Action it is possible to (re) set the name of a project.

This can be used, for example, to define the name according to a naming convention, regardless of the value entered by the user in the "Add project" dialog. However, since the name entered also has a wildcard equivalent, it is also possible to simply expand and change it by using additional wildcards.

The Project Set Name Action can be queried in every stream in which a project object is present.

3.2.1.10. Project Set Path Action

With the help of the Project Set Path Action it is possible to (re) set the file path of a project.

This can be used, for example, to link an existing project object (database / metadata) in helmut with another physical project object.

For example, when restoring a project from a backup, it would be conceivable to simply link the database object with the backup and you will work with the backup the next time the project is opened.

The Project Set Path Action can be queried in every stream in which a project object is present.

3.2.1.11. Project Set Tag Action

With the help of the Project Set Tag Action it is possible to (re) set the template of a project.

This can be used, for example, to dynamically change a tag depending on the work steps that have taken place in order to make projects logically sortable according to these.

The Project Set Tag Action can be queried in every stream in which a project object is present.

3.2.1.12. Project Set Template Action

With the help of the Project Set Template Action it is possible to (re) set the template of a project.

This can be used, for example, to sort projects differently in the view in FX. Once defined, the template is simply a metadata entry that can be called up using a corresponding wildcard in the stream designer. Due to this fact, workflows can be directly dependent on the project template and therefore it might make sense to want to change the template of a project.

The Project Set Template Action can be queried in every stream in which a project object is present.

3.2.1.13. Project as JSON Action

With the help of the project as JSON action, it is possible to write all information that belongs to the project object in a JSON file and to define the location where it should be stored.

This can be necessary for various workflows in which the payload of the stream should be externally accessible. The most important function of the node is to provide this information for the Helmut4 panel. When opening a project, the panel searches for this file and loads all relevant information in order to function smoothly.

3.2.1.14. Project Delete Action

With the help of the Project Delete Action it is possible to delete a project from the database. The ID of the project that is to be deleted must be specified for this.

A conceivable workflow would be, for example, not to delete a project using the Delete button provided for FX, but to build a custom FX stream that is used to delete the project. The node can be used in two different ways. Option 1 provides that the project is deleted from the database and the trigger "delete_project" is executed (via which, for example, the physical folder on the storage could be deleted). The other option provides that this trigger is suppressed. (Option in the node). This makes it possible to build a delete workflow that is completely independent of FX and, for example, could be executed at the end of a Housekeeper workflow.

3.2.1.15. Project from JSON Action

With the help of the Project from JSON Action it is possible to load the entire information of a JSON file into a stream and to use it further. This is especially true for JSON files created by Helmut (Project as JSON Action Node)

A conceivable workflow would be, for example, that the information belonging to a project is written to a JSON file when the project is created, which is stored in a watch folder. For every file that is placed in this watch folder, it would now be possible to load the information from the JSON file into the stream that is executed for this file. This would make it possible, for example, to use the project ID within the stream and to clearly assign the files to a project.

3.2.1.16. Project Import Action

With the help of the Project Import Action it is possible to import a project into Helmut via a workflow (stream) without using the "Add project" dialog. For example via a watch folder. The node itself triggers the streams whose trigger is "create_project" and transfers the parameters required for this. These are:

- name
- File path (path to the project to be imported)
- group
- category
- template
- project type

The Project import Action can be queried in every stream in which a project object is present.

3.2.1.17. Project Import to External Helmut Action

With the help of the Project Import to External Helmut Action, it is possible to import an existing project in Helmut via a stream into another (external) Helmut instance. This node works exactly like the Project Import Action, but also allows the Helmut Server URL and the Client ID to be specified for authentication via OAuth.

3.2.1.18. Project Lock Action

With the help of the Project Lock Action it is possible to lock a project, for example via a stream that is triggered by the "open_project" trigger. This is possible for any type of project in Helmut, and can be used in two different ways.

Simple locking, which indicates that the project has already been opened by a user, and forced locking, in which the project can definitely no longer be opened by other users until it has been unlocked. Unlock can also be carried out by another user, depending on the access rights, or takes place as soon as the project is closed.

A project can also be locked even though it is not open, for example because the project should no longer be edited after a defined process has been completed, etc.

The Project import Action can be queried in every stream in which a project object is present, or if the ProjectID is known.

3.2.1.19. Project Status Update Action

With the help of the Project Status Update Action it is on the one hand possible to set or change the status of a project, on the other hand you can lock or unlock a project at the same time. There are 4 statuses available: Locked, Housekeeper, Archived and Restore. In addition to the regular Project Lock Action, it is possible to make the project unlockable and to display a progress, for example if it is a project that is currently being processed by the housekeeper. The progress can be defined statically between 1 and 100 as well as continuously spinning (-1). Entering -2 in the Progress field removes the progress bar. It is possible to suppress the "unlock_project" event if it is not desired that this should be carried out.

For example, if you want to archive a project, it may be necessary to indicate to the user whether the archiving process has already started or not. Up to the moment when the archiving process starts (job in the dashboard was accepted by a render node) it is logged, but no progress is displayed. Furthermore, a successfully completed archiving process can in turn lead to the status changing to indicate this as well.

3.2.1.20. Project Update Modification Date Action

With the help of the Project Update Modification Date Action, it is possible to set the modification date of a project dynamically via a stream. Usually the date is only changed when the project is closed, duplicated and edited. For example, if you want to change the modification date of a project after an export or import (Cosmo) has taken place, this node can be used.

3.2.1.21. Project Duplicate Action

With the help of the Project Duplicate Action, it is possible to duplicate an existing project within Helmut. The node refers to the project that needs to be duplicated by giving the Project ID of the project that needs to be duplicated. The user can also set the name of the new project and decide if this new project should be served as the active project within the stream.

3.2.2. Job Action Nodes

3.2.2.1. Job Metadata Remove Action

With the help of the Job Metadata Remove Action it is possible to remove a metadata entry that has already been set on an asset. This only applies to custom metadata.

Custom asset metadata is displayed in Cosmo when you click anywhere in the table of displayed assets between the displayed values. See Figure 3.2

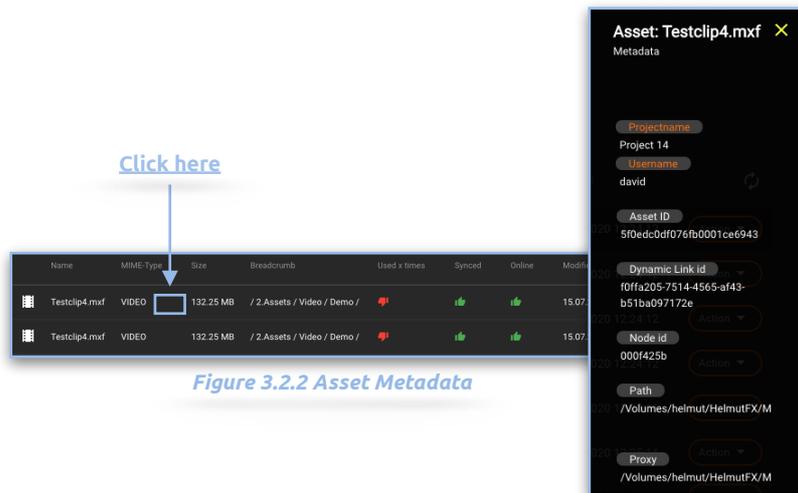


Figure 3.2.2 Asset Metadata

3.2.2.2. Job Metadata Changer Action

With the help of the Job Metadata Changer Action it is possible to change a metadata entry that has already been set on an asset. This only applies to custom metadata.

Custom asset metadata is displayed in Cosmo when you click anywhere in the table of displayed assets between the displayed values. See Figure 3.2

3.2.2.3. Job Priority Action

By default, each job is assigned the priority defined in the profile. With the help of the job priority action it is possible to change this priority again within the executed workflow / job. This can be done, for example, depending on a condition. Example:

If a sequence is exported via the Premiere Pro Panel Extension, each job is given the same priority if the same profile is selected. If you want to prioritise individual jobs differently, for example those of a specific user or those with a specific metadata field, it would be possible to change the priority for this user or for this metadata field via a pre-stream before the job is added to the dashboard's queue .

3.2.2.4. Job QScan Action

With the help of the Job QScan Action it is possible to transfer a file directly to QScan via a stream in order to carry out the necessary QC of this file with a template defined via the node.

For this, the corresponding template must be created in QScan and the storage location of the file that is to be transferred must be made known to QScan beforehand (repository). QScan returns the analysis result in the form of markers and these can be further processed in the further course of the stream (transfer to Cosmo and Premiere). Example workflow:

For example, it would be possible to transfer every exported file to QScan in order to check the result technically. If an error is found (defined search for parameters via QSCAN template possible) this can be used as a condition to change the further course of the stream and to send the user a message.

Another variant would be that an exported file would be transferred to QSCAN ({job.destination} from AME), then checked into COSMO (same project), and thereby automatically reimported into the open project. There the analysis results could be checked directly and any changes made.

3.2.2.5. Job Render AAF in Premiere Action

With the help of the job render AAF in Premiere Action it is possible to create an AAF file of the timeline. This can be done on the system that initiates the job (for example via an export profile) or on a dedicated rendering node. All parameters that Premiere provides are used in the node. The execution of the process addresses the Extend script function of Premiere Pro and is therefore carried out by the invisible Helmut4 Premiere Panel.

3.2.2.6. Job Render in AME Action

With the help of the Job Render in AME Action it is possible to transfer a file to AME and process it with a defined encoding profile. This is equally possible for imports and exports and can be executed on a MAC or Windows client (both support AME).

AME returns the progress of the encoding process to Helmut and this is displayed on the dashboard if the node is used as part of a workflow that creates a job object. This is always the case when the Helmut4 panel (auto import, export) is involved, or a watch folder that is monitored by the server.

The node does not need any information about the file extension in the source path, as this is set by AME itself (is defined in the .epr file). If desired, AME will also count up if the file is already available on the target path. (Setting in AME). The node sets the wildcard {job.destination} in the payload job, which in the further course of the stream contains the target path of the file defined in the node including the file extension.

3.2.2.7. Job Render Proxy in AME Action

The Job Render Proxy in AME Action works exactly like the Job render in AME Action, with the only difference that it defines the target path defined in the node as wildcard {job.proxy} in the job payload. This is now available in the further course of the stream.

This node can be used, for example, to generate a suitable proxy file for the Premiere proxy workflow. This can be transferred to Cosmo and linked to a high res file.

3.2.2.8. Job Render Premiere Action

With the help of the Job Render in Premiere Action it is possible to encode a file or sequence via Premiere Pro. The node works exactly like the job speakers in AME Action, with the difference that Premiere is used as an encoder and the hidden Helmut4 Premiere panel is addressed. It is therefore possible to encode on the workstation on which the job was sent, or on a dedicated render node.

Another difference to Job Render in AME action is that Premiere does not count up during encoding if the file to be encoded is already available at the destination. The file extension must also be specified, even if it is defined in the .epr file.

3.2.2.9. Job Render with FFmpeg Action

With the help of the Job Render with FFmpeg Action it is possible to encode a file using FFmpeg. As it is the case for Adobe Media Encoder, FFmpeg needs to be installed on a supported OS.

FFmpeg returns the progress of the encoding process to Helmut and this is displayed on the dashboard if the node is used as part of a workflow that creates a job object. This is always the case when the Helmut4 panel (auto import) is involved, a watch folder that is monitored by the server or when a web import is triggered.

The path to the FFmpeg executable needs to be set, file extension for the input and destination file need to be included. FFmpeg parameters can be set in the input and output. FFmpeg will not count up if the file is already available on the target path. To count up the file name, the File Increment Name Action should be used together with the FFmpeg node. The path for the output file can be accessed via the wildcard {stream.last_result} in the next node in the stream or anywhere in the stream.

3.2.2.10. Job Start Premiere Action

With the help of the Job Start Premiere Action it is possible to start a certain Premiere version by entering the path to the .exe file (Windows) or .app file (Mac). This node can be used, for example, before the Job Render AFF in Premiere Action to ensure that Premiere has started before the job is submitted to Premiere.

3.2.2.11. Job Status Update Action

With the help of the Job Status Update Action it is possible to change the status and the message of the job that is displayed in the dashboard at any time. This is useful, for example, when an action node is being executed that does not change its status independently. This is only the case if it is a job action. Example:

A sequence is exported and encoded using the "Job Render in AME" action. The encoded file is added to a project in Flow using the Flow Add Assets To Project Action. This process is not shown in the dashboard because Flow does not return the progress and therefore it cannot be displayed.

The job status update action provides a remedy, as it can change the status and also display a progress. The progress can be defined statically between 1-100 or defined as a spinner by entering -1. This spinner continues until the status is changed again via another instance of the node.

The possible statuses are: Queued, Running, Successful and Failed.

Queued is a special feature here, as this status immediately ensures that the job in the dashboard is set back to queued and the job is thus ended for the current render node. This makes it possible to put a job back into the queue under certain circumstances. Example:

If you want to encode a growing file, but only do this when the file is no longer growing, a solution would be to use the Growing File Condition to check whether the file is still growing. If this is the case, the Job Status Update Action can be used to put the job back into the queue. As soon as this job is fetched from the next free render node, it starts over with the job. Only when the file stops growing does the stream's behavior change and the file is encoded.

3.2.2.12. Job as JSON Action

With the help of the Job as JSON action, it is possible to write all information that belongs to the job object in a JSON file and to define the location where it should be stored.

This can be necessary for various workflows in which the payload of the stream should be externally accessible.

3.2.2.13. Job from JSON Action

With the help of the Job as JSON action, it is possible to read a job JSON file and paste it into the job. In combination of the [Job as JSON](#) and Job from JSON node delayed jobs can be created. Those can be collected and processed later at any given time e.g. by a watchfolder that is triggered by cron. The node does set up

3.2.2.14. Job Set Project ID Action

With the help of the Job Set Project ID Action, it is possible to add the ID of a project in the stream to the job (job object), provided this is known. This is especially necessary when jobs are created using a watch folder. In this case there is usually no project ID and the job would not be visible to a normal user in the panel or on the web. If this works, the ID must already be set in the pre-stream of a watch or job. This means it is available in the following stream and can be displayed correctly. The ID of a project can be extracted from the project overview in FX (hidden overlay) or from a JSON file that was previously written.

3.2.2.15. Job Create Job Action

With the help of the Job Create Job Action, it is possible to create a job via a stream instead of creating it via a regular route (auto import, web import, export, etc.). It is possible to use the profile ID, the job name, the Define priority as well as the rendering node(s) for which the job is intended. In addition, metadata and stream variables can be adopted from the stream that creates the job. The profile ID of a profile can be found in the hidden overlay of the respective profile. This node can be used for a variety of purposes.

Example 1:

If you want to automatically copy the assets in a project to the central storage after importing the project, it is possible to index the project via a stream attached to the event "post_create_project" and to create a job that uses a Housekeeper profile via which the assets in the project are copied. As a result, this process is detached from the actual import process and can be transferred to dedicated nodes.

Example 2:

If you do not want the editor to leave the FX interface in order to trigger Housekeeper workflows, for example, a possible solution would be to create custom FX streams, which are used to create jobs in which Housekeeper profiles are used.

3.2.2.16. Job Delete Action

With the help of the Job Delete Action, it is possible to delete a job using its ID via a stream. There are actually only a few use cases for this, but one makes perfect sense. For example, if you want to use a pre-stream to define whether an asset should be processed at all or not (e.g. auto import), the job can simply be removed again depending on the condition. Since PreStreams are only visible to administrators, only those jobs that are actually processed would appear in the user's dashboard.

3.2.2.17. Job Execute Extendscript in Premiere Action

With the help of the Job Execute Extendscript in Premiere Action it is possible to execute every Extendscript call that Premiere makes available. In order for this to work, the render node that executes the stream containing this node must have at least Premiere installed or opened as recommended. The node described here communicates with the invisible Helmut4 Premiere Panel Extension (installed automatically with the client), which executes the Extendscript call.

3.2.2.18. Job File Copy Action

With the help of the Job File Copy Action it is possible to copy a file from one path to another. It shows the progress of the copy process in the dashboard.

3.2.2.19. Job Folder Copy Action

With the help of the Job Folder Copy Action it is possible to copy a folder including its contents from "a" to "b". This node is mainly intended to copy folders that contain video assets. It is recommended to use the Job Folder Copy Action in an asynchronous way of the stream, as this allows the progress of the copy process to be displayed in the dashboard. (requires HIO)

3.2.2.20. Job RClone Copy Action

With the help of the job RClone Copy Action it is possible to copy assets or folders to a remote location. In order for this to be possible, RClone must be installed on the workstation on which this node (within a stream) is running. The path to the Executbale is given in the node, but can also be freely defined. The configuration of the remote location must be done via RClone itself. See the documentation for RClone. The name of the remote location can be defined when creating the remote location in RClone and must be specified in the node.

Example:

If you want to copy a file to a remote location with the name "Remote-Location" and store it there in a folder with the name: "Folder names", the target directory must be specified in the following way:

Remote location: / folder names

3.2.2.21. Job RClone Mkdir Action

With the help of the Job RClone Mkdir Action it is possible to create a folder on a remote location. In order for this to be possible, RClone must be installed on the workstation on which this node (within a stream) is running. The path to the Executbale is given in the node, but can also be freely defined. The configuration of the remote location must be done via RClone itself. See the documentation for RClone. The name of the remote location can be defined when creating the remote location in RClone and must be specified in the node.

Example:

If you want a folder with the name: "Folder name" on a remote location with the name "Remote-Location", the target directory must be specified in the following way:

Remote location: / folder names

3.2.2.22. Job RClone Move Action

With the help of the job RClone Move Action it is possible to move assets or folders between remote locations. In order for this to be possible, RClone must be installed on the workstation on which this node (within a stream) is running. The path to the Executbale is given in the node, but can also be freely defined. The configuration of the remote location(s) must be done via RClone itself. See the documentation for RClone. The names of the remote location(s) can be defined when creating the remote location(s) in RClone and must be specified in the node.

Example:

If you want to move a file from a remote location called: "Remote-Location1" to a remote location called "Remote-Location2" the source path and destination path must be specified in the following way:

Source path: Remote-Location1: / folder_name/name_of_the_clip.mxf

Destination path: Remote-Location2:/folder_name

3.2.2.23. Job RClone Purge Action

With the help of the job RClone Purge Action it is possible to delete a file or a folder from a remote location. In order for this to be possible, RClone must be installed on the workstation on which this node (within a stream) is running. The path to the Executable is given in the node, but can also be freely defined. The configuration of the remote location must be done via RClone itself. See the documentation for RClone. The name of the remote location can be defined when creating the remote location in RClone and must be specified in the node

Example:

If you want to delete a folder with the name: "Folder names" from a remote location with the name "Remote-Location", the target path must be specified as follows:

Remote location: / folder names

3.2.2.24. Job RClone Rmdir Action

With the help of the job RClone Rmdir Action it is possible to remove a folder (directory) from a remote location if it is empty. In order for this to be possible, RClone must be installed on the workstation on which this node (within a stream) is running. The path to the Executable is given in the node, but can also be freely defined. The configuration of the remote location must be done via RClone itself. See the documentation for RClone. The name of the remote location can be defined when creating the remote location in RClone and must be specified in the node.

Example:

If you want to remove a directory with the name: "Directory name" from a remote location with the name "Remote-Location", the target path must be specified as follows:

Remote location: / Directory name

3.2.2.25. Job Export FCP XML in Premiere

With the help of the Job Export FCP XML in Premiere Action node it is possible to create a new project to export a Final Cut Pro XML in Premiere (panel in Premiere required). You can export a sequence into an interchangeable XML format.

3.2.2.26. Job Create Split Jobs Action (BETA)

With the help of the Job Create Split Jobs Action it is possible to split (divide) the job of a profile into a desired number of single split jobs. This only works for Premiere Pro sequences. Doing this the sequence job will be divided into equal parts of same length. On top of that the user is able to consider In and Out markers in the sequence and set the priority for the queued jobs in Premiere. To save time during the rendering process every split job can be executed on a dedicated single render node.

To enable the Split & Stitch Workflow it is necessary to add the Job Stitch Splits Action Node at the end of the stream, that is part of the profile that serves as the profile for the splitted jobs.

Note: As a BETA Version this will only be supposed for sequences with 25fps. When creating streams with Job Create Split Jobs Action it is mandatory to use it within a Split & Stitch workflow also using Job Stitch Splits Action.

3.2.2.27. Job Stitch Splits Action (BETA)

With the help of the Job Stitch Splits Action it is possible to connect the exported files from Job Create Split Jobs Action by putting them back together in sequentiell order relating to one certain footage. For this a separate job will be send to AME and the file can be stored in a given destination.

Example:

A huge render job will be divided into 5 split jobs. A profile to split the job will be created and set as Prestream. It will be defined that the jobs should be executed on 2 existing render nodes. After the jobs have been finished AME will put the results together in a final export.

3.2.3. Premiere Action Nodes

3.2.3.1. Premiere Generate UUID Action

With the help of the Premiere Generate UUID Action, it is possible to create a Unique Premiere Project ID and use it for a newly created project. All you have to do is enter the Helmut4 Project ID in the node. Helmut4 creates a new ID and uses it automatically for the project.

3.2.3.2. Premiere Alert Dialog Action

With the help of the Premiere Alert Dialog Action it is possible to send a message to a user in the Helmut4 panel. This message is sent to the hidden Premiere Pro panel and uses an extendscript function provided by Premiere Pro. This means that it is only possible to send a message to a user if he triggers an action via the Helmut4 panel and is also the executing user. For example during the export. If the export is to be carried out on a dedicated rendering machine (setting in the profile), this is of course not possible, since the user who accepts the job is then not the user who submitted the job.

3.2.3.3. Premiere Confirm Dialog Action

With the help of the Premiere Confirm Dialog Action, a message that the user must confirm can be sent via the Helmut4 panel. If the message is confirmed with "YES", the node is successful. If the message is canceled with "NO", the node will fail. Optionally, it can be defined that the node is successful if the user ignores the message.

The message is sent to the hidden Premiere Pro control panel and uses an Extendscript feature provided by Premiere Pro. This means that it is only possible to send a message to a user if he triggers an action via the Helmut4 control panel and is the executing user at the same time. For example when exporting. If the export is to be carried out on a dedicated rendering computer (setting in the profile), this is of course not possible, since the user who accepts the job is then not the user who sent the job.

3.2.3.4. Premiere Force Native Lock Action

The Premiere Force Native Lock Action ensures that Native Premiere Locking is activated within the project file. To do this, the node writes the user name in the path of the user profile on the workstation.

3.2.3.5. Premiere Native Lock Action

A Premiere Project can be locked using the Premiere Native Lock Action. Here, Premiere's own native locking is used and the corresponding lock file is written in the path that was defined via the node. Depending on whether the stream in which this node is used is a server or a client stream, the server or the client monitors this file. As long as this file exists and can be accessed by the server or the client, the corresponding project is blocked in the HelmutFX project overview. Once the project is closed, Premiere Pro deletes this file, which unlocks the project.

3.2.3.6. Premiere OS Path Mapper Action

The Premiere OS Path Mapper Action ensures that all paths within a project are mapped correctly, depending on the operating system installed on the workstation that opens the premiere project. This node is necessary if several operating systems are used in the editing environment and you want to avoid manually linking the files in the project.

3.2.3.7. Premiere Open Choose Dialog Action

With the help of the Premiere Open Choose Dialog Action, a Finder or Explorer window can be opened that shows a path defined in the node. It can be defined whether an entire folder or one or more files can be imported.

This function uses Extendscript and can therefore only be used if Premiere is opened and the stream in which this node is used is executed by the client on which the corresponding project is opened.

3.2.3.8. Premiere Path Settings Action

With the help of the Premiere Path Settings Action, all paths that can be defined in the Project Settings of Premiere Pro can be set. This can be done while creating a project or while opening it. No path is set by default, but the auto save location template is the one that should be used to use the HelmutFX "restore saves" function. HelmutFX expects all Auto Save Files to be saved in this folder. Every other path can be set as desired.

3.2.3.9. Premiere Prompt Dialog Action

With the help of the Premiere Prompt Dialog Action, a message can be sent to a user asking him to enter a value. If the message is confirmed with "YES", the node is successful. If the message is canceled with "NO", the node will fail. The value to be entered by the user can be saved by default, but can also be overwritten. This value is available in the stream via the wildcard {stream.last_result}.

The message is sent to the hidden Premiere Pro control panel and uses an Extendscript feature provided by Premiere Pro. This means that it is only possible to send a message to a user if he triggers an action via the Helmut4 control panel and is the executing user at the same time e.g. when exporting. If the export is to be carried out on a dedicated rendering computer (setting in the profile), this is of course not possible, since the user who accepts the job is then not the user who sent the job.

3.2.3.10. Premiere Start Helmut Panel Action

With the help of the Premiere Start Helmut Panel Action it is possible to start the Helmut Panel in Premiere in case it is not already opened. The analogue way is to start the window manually via Premiere Extensions.

3.2.3.11. Premiere Version Converter Action

With the help of the Premiere Version Converter Action it is possible to convert every Premiere project opened via HelmutFX into the desired version number. This happens fully automatically if the node is used in an "open_project" stream, for example. It is not recommended to convert projects that were created on a newer version into an older version, as this can lead to problems and corrupt projects due to the fact that functions from newer versions may be missing in older versions.

3.2.4. Cosmo Action Nodes

3.2.4.1. Cosmo Add Asset To Project Action

With the help of the Cosmo Add Asset to Project Action it is possible to add an asset to one or more projects that exist in Cosmo via a stream. For that the project ID's of considered projects will be added to field 'Project ID List' in comma separated order. A distinction is made between asset path and asset name. The asset path is the actual storage path including name and file extension, the asset name is the name the asset should have in the database and thus in Cosmo, Premiere or After Effects. A breadcrumb can be specified within the node. It can also be filled via a wildcard {job.breadcrumb} depending on which trigger was used for the stream. In the case of an auto import or a web upload, this wildcard is available, but not in the case of an export or watch folder workflow. There is an option to keep the original asset name (check box) to ensure, for example, within an auto import that the assets are not renamed during synchronization via the panel. This enables the editors to rename the assets in premiere during an auto import process and to keep these names. The set names can be updated by re-indexing the project in the Cosmo database. There is an option to keep the original breadcrumb (check box) to ensure, for example, within an auto import that the assets are not moved during synchronization via the panel. This enables the editors to change the location of the assets in premiere during an auto import process and to keep these locations.

3.2.4.2. Cosmo Add Info To Sequence Action

With the help of the Cosmo Add Info to Sequence Action it is possible to write metadata and markers that are present in the stream back to the sequence instead of applying this information to the exported file. To do this, the node is simply inserted in the stream at the desired location. Example:

If a sequence is encoded using the job render in AME Action and then transferred to QScan, the information found by QScan is returned to the stream as a marker. This information can now be transferred to the sequence that served as the source for the export. This makes it possible to check the result of the quality control by QScan directly on the timeline.

3.2.4.3. Cosmo Add Proxy To Project Action

With the help of the Cosmo Add Proxy to Project Action it is possible to add or update a proxy to an existing HighRes asset in Cosmo. If the HighRes asset has not been added before the proxy has been added, the node will fail. There are three options of attaching that depending on the input parameters which the user provides. "Asset ID" has the highest priority and gives the option to attach a proxy to one individual asset. "Project ID" means that all assets with the same Highres Path inside the project will receive the proxy. Providing only the „Highres Path“ without the other two options forces all assets to be saved in the database that match with HighresPath to receive the proxy. With the "Auto Sync" option it can be decided if the proxy should automatically be synced whenever the project will be opened next time or immediately be synced if the project is already opened. Otherwise it can be synced manually with the sync icon in the Helmut4 panel.

3.2.4.4. Cosmo Change Asset Action

With the help of the Cosmo Change Asset Action it is possible to change the path of a file in Cosmo globally. For example, if a file is moved (archive) via a housekeeper workflow, the new storage location can be made known and, if the autosync function is activated in the node, it can also be automatically reset in all premiere projects (after the project has been opened).

3.2.4.5. Cosmo Get Project Assets Action

With the help of the Cosmo Get Project Assets Action it is possible to extract the assets checked into a project in Cosmo individually via a stream. In the node, you can define which assets you want to process by making the following selection:

- Exists only in this project:
 - If this checkbox is checked, all assets that are available in the project are checked to see whether they are still available in at least one other project. If the checkbox is off, all assets will be extracted.
- Filter - Both:
 - All assets are extracted
- Filter - Used in timeline
 - Only the assets that were used on the timeline (s) are extracted
- Filter - Not used in timeline:
 - All assets that are in the project and have not been used on any timeline are extracted.

The node iterates until all extracted assets have been processed. If any stream fails while processing an asset, the node fails as a whole. The node result will be {job.assetId}. This node is the key to the efficient management of project-based assets, as all assets can be checked and extracted depending on the existing projects.

3.2.4.6. Cosmo Change Project Asset Action

With the help of the Cosmo Change Project Asset Action it is possible to change the path of a file in Cosmo just for a specific project. For example, if a file is moved (archive) via a housekeeper workflow, the new storage location can be made known and, if the autosync function is activated in the node, it can also be automatically reset in the corresponding premiere project (after the project has been opened).

3.2.4.7. Cosmo Project File Index Action

With the help of the Project File Index Action, it is possible to index a project file (Premiere / After Effects). It is recommended to only do this when the project is closed. The indexing extracts all information from the project. These include:

- Folder structures and folder names
- Files and metadata
- Sequences

All changes that are detected during indexing compared to the status in Cosmo are changed in Cosmo. It is possible to define that the "indexed_assets" trigger is triggered for newly found assets and the information on these assets is transferred.

In addition, the "unindexed_assets" trigger is triggered for assets that have been removed from the project. Within a stream that is triggered by this trigger, it is possible to check whether these removed assets are still available in other projects or not. Every asset that is removed from a project and is no longer available in any other project is recorded in the database as an orphan clip, but is no longer accessible.

Within the node it is possible to define advanced indexing for the project file. If checked, the following information about sequences will also be extracted:

- Source and destination time code of all assets on the timeline
- Track that was used per asset
- Metadata of the assets

This information is not visualized in Cosmo but is available in the database.

3.2.4.8. Cosmo Get Project Sequences Action

The node Cosmo Get Project Sequence Action iterates until all extracted sequences of a project have been processed. If any stream fails while processing an asset, the node fails as a whole. The node result will be {job.assetId}. This node is the key to the efficient management of project-based assets, as all assets can be checked and extracted depending on the existing projects.

3.2.4.9. Cosmo Asset Report Action

The node Cosmo Asset Report Action will export a JSON report detailing in how many projects the asset has been used so far. By giving the asset ID and the destination file it can also be defined if the file will be appended, incremented or deleted in case of a conflict. It is designed and recommended to use this node in cooperation with Cosmo Get Project Assets Action to report for a proceeded asset.

3.2.4.10. Cosmo Sequence Report Action

The node Cosmo Sequenz Report Action will export a JSON report detailing all assets used in the sequence while the corresponding project must have been indexed with the 'advanced' option. By giving the sequence (asset) ID and the destination file it can also be defined if the file will be appended, incremented or deleted in case of a conflict. It is designed and recommended to use this node in cooperation with Cosmo Get Project Assets Action to report for a proceeded sequence.

3.2.4.11. Cosmo Change Asset Metadata Action

With the help of Cosmo Change Asset Metadata Action it is possible to manage custom metadata besides typical job and project metadata. This node will add or update one or multiple metadata key value pairs to an asset by referring to the asset ID. It allows users to change asset metadata of already indexed Assets. The corresponding metadata entries will be added (new entries) to or updated (existing entries) in the database (MongoDB - assetDB) into the array „customMetadata“. Avoid using this node right after the Cosmo Add Asset To Project Action node or at least provide the Sleep Action in between and wait for 4000+ ms.

3.2.4.12. Cosmo Delete Asset Metadata Action

With the help of Cosmo Delete Asset Metadata Action it is possible to delete one or multiple custom metadata besides entries of already indexed assets. In comparison to Cosmo Change Asset Metadata Action this node removes particular asset metadata by metadata key(s) by asset ID. The node ignores non existing keys and continues with the successful path in this case. Avoid using this node right after the Cosmo Add Asset To Project Action node or at least provide the Sleep Action in between and wait for 4000+ ms.

3.2.4.13. Cosmo Get Asset Metadata Action

With the help of Cosmo Get Asset Metadata Action it is possible to request asset metadata and save it in the wildcard {node.result.?}. Metadata values can be retrieved with 'nodeID_metadataKey' e.g. 'd7d99d44-07a1-40f8-8053-3db89482342c_metaKey'. Characters as {} and % are not supported for metaKey. For the use of this node an asset with existing asset metadata in Cosmo is required and the asset ID as well as the metadata key must be known. Avoid using this node right after the Cosmo Add Asset To Project Action node or at least provide the Sleep Action in between and wait for 4000+ ms.

Example:

Consider an asset with the asset ID "6225d94587a52b082de53896", the metadata key "type" and value "VIDEO". To retrieve the value "VIDEO" a Cosmo Get Asset Metadata Action node must be added into the stream with 6225d94587a52b082de53896 as asset ID beforehand.

The node ID e.g. "d7d99d44-07a1-40f8-8053-3db89482342c" of this node can be used to get the metadata value "VIDEO" for its metadata key by using the {node.result.?} wildcard as following: {node.result.d7d99d44-07a1-40f8-8053-3db89482342c_type}.

3.2.5. After Effects Action Nodes

3.2.5.1. After Effects OS Path Mapper Action

The After Effects OS Path Mapper Action ensures that all paths within a project are mapped correctly, depending on the operating system installed on the workstation that opens the after effects project. This node is necessary if several operating systems are used in the editing environment and you want to avoid manually linking the files in the project.

3.2.5.2. After Effects Alert Dialog Action

With the help of the After Effects Alert Dialog Action it is possible to send a message to a user in the Helmut4 panel. This message is sent to the hidden After Effects panel and uses an Extendscript function provided by After Effects. This means that it is only possible to send a message to a user if he triggers an action via the Helmut4 panel and is also the executing user. For example during the export. If the export is to be carried out on a dedicated rendering machine (setting in the profile), this is of course not possible, since the user who accepts the job is then not the user who submitted the job.

3.2.5.3. After Effects Open Choose Dialog Action

With the help of the After Effects Open Choose Dialog Action, a Finder or Explorer window can be opened that shows a path defined in the node. It can be defined whether an entire folder or one or more files can be imported.

This function uses Extendscript and can therefore only be used if After Effects is opened and the stream in which this node is used is executed by the client on which the corresponding project is opened.

3.2.5.4. After Effects Prompt Dialog Action

With the help of the After Effects Prompt Dialog Action a message can be sent to a user asking him to enter a value. If the message is confirmed with "YES", the node is successful. If the message is canceled with "NO", the node will fail. The value to be entered by the user can be saved by default, but can also be overwritten. This value is available in the stream via the wildcard {stream.last_result}.

The message is sent to the hidden After Effects control panel and uses an Extendscript feature provided by After Effects. This means that it is only possible to send a message to a user if he triggers an action via the Helmut4 control panel and is the executing user at the same time e.g. when exporting. If the export is to be carried out on a dedicated rendering computer (setting in the profile), this is of course not possible, since the user who accepts the job is then not the user who sent the job.

3.2.5.5. After Effects Start Helmut Panel Action

With the help of the After Effects Start Helmut Panel Action it is possible to start the Helmut Panel in After Effects in case it is not already opened. The analogue way is to start the window manually via After Effects Extensions.

3.2.5.6. Job Execute ExtendScript In After Effects Action

With the help of the Job Execute Extendscript in After Effects it is possible to execute every Extendscript call that After Effects makes available. In order for this to work, the render node that executes the stream containing this node must have at least After Effects installed or opened as recommended. The node described here communicates with the invisible Helmut4 After Effects Panel Extension (installed automatically with the client), which executes the Extendscript call.

3.2.5.7. Job Start After Effects Action

With the help of the Job Start After Effects Action it is possible to start a certain After Effects version by entering the path to the .exe file (Windows) or .app file (Mac).

3.2.6. Third Party Action Nodes

3.2.6.1. Editshare Set ACL Action

With the help of the Edit Share Set ACL Action it is possible to set the ACL rights for a path (folder or file), for one or more users, or for one or more groups. To do this, the corresponding user or group must have access rights to the drive.

Communication with the Edit Share API takes place via the Edit Share Admin Account, which must be entered under HelmutFX -> Preferences -> Modules -> Edit Share Module.

The user who executes the workflow does not need ACL full control rights on the corresponding volume.

3.2.6.2. Editshare Delete ACL Action

With the help of the Edit Share Delete ACL Action it is possible to remove the ACL rights for a path (folder or file), for one or more users, or one or more groups. To do this, the relevant user or group must have access rights to the drive.

Communication with the Edit Share API takes place via the Edit Share Admin Account, which must be entered under HelmutFX -> Preferences -> Modules -> Edit Share Module.

The user who executes the workflow does not need ACL full control rights on the corresponding volume.

3.2.6.3. EFS Create User Action

With the help of the EFS Create User Action it is possible to add a user to an Editshare EFS System. The User name and user password has to be specified within the node. It's only possible to add one user at a time.

3.2.6.4. EFS Delete User Action

With the help of the EFS Delete Users Action it is possible to delete one or multiple users from an Editshare EFS System. The input field for username(s) accepts a comma separated list.

3.2.6.5. EFS Add Users To Group Action

With the help of the EFS Add Users to Group Action it is possible to add one or multiple existing users to an existing group within an Editshare EFS System. The group name and usernames must be specified within the node. The input field(s) for username(s) accept a comma separated list.

3.2.6.6. EFS Remove User From Group Action

With the help of the EFS Remove Users from Group Action it is possible to remove one or multiple existing users from an existing group within an Editshare EFS System. The group name and usernames must be specified within the node. The input field(s) for username(s) accept a comma separated list.

3.2.6.7. EFS Add Users To Media Space Action

With the help of the EFS Add Users to Media Space Action it is possible to add one or multiple existing users to an existing media space within an Editshare EFS System. The media space name and usernames must be specified within the node. The input field(s) for username(s) accept a comma separated list.

3.2.6.8. EFS Remove Users From Media Space Action

With the help of the EFS Remove Users From Media Space Action it is possible to remove one or multiple existing users from an existing media space within an Editshare EFS System. The media space name and usernames must be specified within the node. The input filed(s) for username(s) accept a comma separated list.

3.2.6.9. EFS Create Group Action

With the help of the EFS Create Group Action it is possible to add one or multiple groups to an Editshare EFS System. The group name(s) must be specified within the node. The input filed(s) for username(s) accept a comma separated list.

3.2.6.10. EFS Remove Group Action

With the help of the EFS Delete Groups Action it is possible to delete one or multiple groups from an Editshare EFS System. The input field for groupname(s) accepts a comma separated list.

3.2.6.11. EFS Add Group To Media Space Action

With the help of the EFS Add Groups to Media Space Action it is possible to add one or multiple existing groups to an existing media space within an Editshare EFS System. The media space name and group names must be specified within the node. The input filed(s) for group name(s) accept a comma separated list.

3.2.6.12. EFS Remove Group From Media Space Action

With the help of the EFS Remove Groups From Media Space Action it is possible to remove one or multiple existing groups from an existing media space within an Editshare EFS System. The media space name and group names must be specified within the node. The input filed(s) for group name(s) accept a comma separated list.

3.2.6.13. EFS Create Media Space Action

With the help of the EFS Create Media Space Action it is possible to add a media space to an Editshare EFS System. The media space name as well as the Quote must be specified. It's only possible to add one media space at a time.

3.2.6.14. EFS Delete Media Space Action

With the help of the EFS Delete Media Space Action it is possible to add a media space to an Editshare EFS System. The media space name must be specified. It's only possible to delete one media space at a time.

3.2.6.15. EFS Update Media Space Action

With the help of the EFS Update Media Space Action it is possible to change a media space on an Editshare EFS System. The media space name must be specified. It's only possible to update one media space at a time.

3.2.6.16. EFS Get Media Space DATA Action

With the help of the EFS Get Media Space data Action it is possible to get all data from a media space on an Editshare EFS System. The media space name must be specified and the data is available in form of JSON payload inside of the stream. To extract a parameter, please use the JSON Extract Action and use {stream.last.result} or {node.result.?.} To load the payload from this node into the JSON Extract Node.

3.2.6.17. EFS Get Free Space Of Media Space Action

With the help of the EFS Get Free Space of Media Space Action it is possible to get the remaining space (in GB) of a media space on an Editshare EFS System. The media space name must be specified and the data is available within the payload inside of the stream. To use this, please use {stream.last.result} or {node.result.?.}

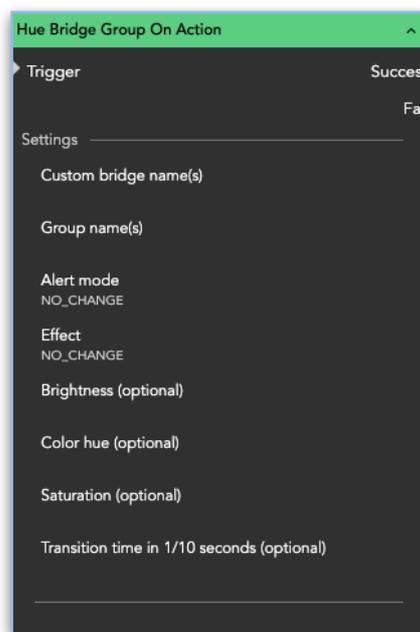
3.2.6.18. Hue Bridge Group On Action

With the help of the Hue Bridge Group On Action it is possible to switch on one or more groups in the Hue system. It is possible to work with several bridges at the same time. It is possible to set a value for brightness, color and saturation of all lamps from all selected groups. These values are optional. If they are not set, the lamps start with the value they had before they were last switched off.

In addition, it is possible to define a value between 1 and 10 seconds for the transition time as the time in which the lamps reach the desired brightness.

As a little gimmick, it is possible to let the lamps flash briefly before switching them on (alert mode) or to run a color loop (effect). The color loop is only supported by Hue Color lamps. Two modes are available for the alert mode: Select (1x flashing) and LSelect (15x flashing).

The node fails if the name of a bridge or the name of a group is misspelled. If one or more group (s) cannot be reached, likewise, but the lamps that were reachable are still switched on.

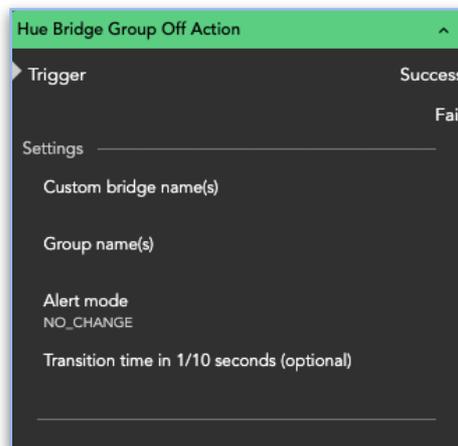


3.2.6.19. Hue Bridge Group Change Action

The Hue Bridge Group Change Action works exactly like the Hue Bridge Group On Action with the simple difference that the lamps assigned to the respective groups are only changed. If a lamp does not support a set parameter, nothing happens. This is because Philips offers 3 different types of lamps: White, White Ambiance and Color. Please ask the manufacturer about the type of light source you are using.

3.2.6.20. Hue Bridge Group Off Action

With the help of the Hue Bridge Group Off Action it is possible to switch off one or more groups in the Hue System. It is possible to switch off one or more groups. The transition time can be set in seconds from 1 to 10 and, if necessary, the lamps can be made to flash via the alert mode before they are switched off.



3.2.6.21. Hue Bridge Lamp On Action

The Hue Bridge Lamp On Action works just like the Hue Bridge Group On Action with the only difference that it is possible to control individual light sources in a targeted manner. The possible parameters and the behaviour of the node are otherwise identical.

3.2.6.22. Hue Bridge Lamp Change Action

The Hue Bridge Lamp Change Action works just like the Hue Bridge Group Change Action with the only difference that it is possible to control individual light sources in a targeted manner. The possible parameters and the behaviour of the node are otherwise identical.

3.2.6.23. Hue Bridge Lamp Off Action

The Hue Bridge Lamp Off Action works just like the Hue Bridge Group Off Action with the only difference that it is possible to control individual light sources in a targeted manner. The possible parameters and the behaviour of the node are otherwise identical.

3.2.6.24. CatDV Add Asset to Catalog Action

With the help of the CatDV Add Asset To Catalog Action it is possible to check in an asset in CatDV and make it available in a certain group. If the group names from Helmut are to be used, these group names must first be created using the CatDV Create Catalog Path action. Groups in Helmut4 correspond to catalogs in CatDV. In addition, a metadata group should exist, as this can be used to control what happens to the asset within CatDV when it has been checked in there.

3.2.6.25. CatDV Create Catalog Path Action

With the help of the CatDV Create Catalog Action it is possible to automatically create a catalog in CatDV. For this purpose the API of CatDV is addressed and the corresponding catalog is created. It is possible to define a folder structure for the catalog, which is also created directly.

3.2.6.26. CatDV Delete Catalog Path Action

With the help of the CatDV Delete Catalog Action it is possible to completely or partially delete an existing catalog in CatDV. This means that it is possible, for example, to remove individual folders from the catalog or the entire catalog.

3.2.6.27. Flow Add Asset To Project Action

With the help of the Flow Add Asset To Project Action it is possible to check in an asset in Flow and make it available in a certain project. If the project names from Helmut are to be used, these project names must first be created using the Flow Create Path action. Project names in Helmut4 can correspond to project names in Flow.

It is possible to transfer markers and metadata, as well as to initiate the flow internal proxy creation for the added asset.

For example this node can always be used if an asset that is loaded into Cosmo or directly into Premiere should also be available in Flow.

3.2.6.28. Flow Create Path Action

With the help of the Flow Create Path Action it is possible to create a project and the folder structure required for this project in Flow. Thus, for example, whenever a project is created in Helmut, a corresponding project can be created in Flow.

3.2.6.29. Flow Delete Path Action

With the help of the Flow Delete Path Action it is possible to delete a folder from a Flow project or the entire project automatically.

3.2.6.30. Flow Toggle Private Project Action

With the help of the Flow Toggle Private Project Action it is possible in Flow to define a project as a private project (only visible to the owner / creator) and to add individual users to this project. This node is helpful if it is not desired that every flow user can see every project, because the Flow Create Path Action creates every project as a public project.

3.2.6.31. SwatIO Job Upload Action

With the help of the Job SwatIO Upload Action it is possible to create a post on Swat.io including a video file upload. There is the option of creating a post draft in preparation (PREPARE), providing a post for scheduled publication (PUBLISH) or instantly publishing it on the desired platform (PUBLISH_NOW).

For using this stream or workflow it is mandatory to insert a valid file path of a video with max. file size of 1GiB. To operate the node it is necessary to enter the corresponding „Channel ID“, which refers to the target channel on Swat.io to assign this post. This Channel ID can be retrieved in the Swat.io Social Media Manager under Channels in the respective URL of the channel.

In addition, the title of the published article and a descriptive text can be defined in the field „Post title“ and „Post message“. Both can be filled by using metadata optionally. Furthermore the „Publication date“ can be added within the corresponding field in the supported format YYYY-MM-DD hh:mm (based on server timezone) for the post actions PREPARE and PUBLISH. Also for that the date and time can be provided as a metadata or wildcard. The field „JWT token“ in the node settings is optional and can be inserted in this field or in the Helmut preferences within the enabled „SwatIO module“.

A possible use case is the publication of posts on the common social media platforms directly via the Helmut4 panel in the form of an export stream.

3.2.6.32. Medialoopster Add Asset To Project Action

With the help of the Medialoopster Add Asset To Project Action, it is possible to add an asset to a project within Media Loopster. To do this, it is necessary to know the project-ID of the corresponding project in Medialoopster. This ID is available if the project was created by Helmut4 and can be called up via the wildcard: {project.custom.?} In which the question mark must be replaced by medialoopster. {project.custom.medialoopster}.

When an asset is added to Medialoopster, the asset ID assigned by Medialoopster is written into Helmut4's database and is available via the wildcard {job.custom.?}. In the wildcard, the question mark must be replaced by medialoopster.—> {job.custom.medialoopster}

3.2.6.33. Medialoopster Create Project Action

With the help of the Medialoopster Create Project Action, it is possible to automatically create a project in Media Loopster. For this purpose the API of Media Loopster is addressed and the corresponding project is created. During this process, the project id of the project that is created in medialoopster is written to helmut's database and is available there as a custom project id. It is possible to define a folder structure for the catalog, which is also created directly. If the node is used to duplicate a project, it is possible to add the source project name to the "Copy of" field, to tell Media Loopster to create another linked instance of each asset (belonging to the source project) within the duplicated project. While doing this, it is possible to share the linked assets or not by using the toggle switch.

3.2.6.34. Medialoopster Delete Asset From Project Action

With the help of the MediaLoopster Delete Asset From Project Action, it is possible to delete (unlink) an asset from a project in MediaLoopster. To do this it is necessary to know both the custom project id and the custom asset id of the project and the asset in medialoopster.

3.2.6.35. Medialoopster Delete Project Action

With the help of the Medialoopster Delete Project Action it is possible to delete an existing Project from Medialoopster. Media Loopster takes care of the corresponding assets within the project. If there is a protection on a single asset that belongs to the project, the project will not be deleted, unless all assets belonging to the project have been deleted.

3.2.6.36. Medialoopster Update Project Delete Date Action

With the help of the Medialoopster Update Project Delete date Action it is possible to change the project delete date within Medialoopster. This can, for example, be done with a stream, attached to the edit_project trigger.

3.2.6.37. Mount EFS Volume (MacOS) Action

With the help of the Mount EFS Volume action it is possible to mount a Editshare EFS media space on a MacOS workstation. This action can, for example, be used to mount a given Editshare media space when the user logs in, using the connected trigger, or when a user open a project, using the open_project trigger.

3.2.6.38. Mount EFS Volume (Windows) Action

With the help of the Mount EFS Volume action it is possible to mount a Editshare EFS media space on a Windows workstation. This action can, for example, be used to mount a given Editshare media space

when the user logs in, using the connected trigger, or when a user open a project, using the open_project trigger.

3.2.6.39. Stratus Create Asset Action

With the help of the Stratus Create Asset Action it is possible to create a placeholder within Stratus. This placeholder can be a clip, a document, a project or generic. In addition, a rating must be given and optional tags can be set. In addition to that the „Description“ field does provide carriage return or new lines by pressing SHIFT + ENTER on the keyboard.

3.2.6.40. Stratus Create Project Action

With the help of the Stratus Create Project Action it is possible to create a project in Stratus. The projectID from Helmut, as well as the path, name and project type can be transferred to Stratus. This node is useful, for example, to transfer projects that are created via Helmut directly to Stratus via a create_project stream.

3.2.6.41. Stratus Generate Unique ID Action

With the help of the Stratus Generate Unique ID Action it is possible to request a unique ID from Stratus in order to use it as a project ID in Helmut. This means that comparisons between the projects can be carried out consistently.

3.2.6.42. Stratus Patch Asset Action

With the help of the Stratus Patch Asset Action it is possible to patch an existing placeholder (clip) and to change individual values (update).

3.2.6.43. Stratus Project Indexed Action

With the help of the Stratus Project Indexed Action it is possible to transfer this information to Stratus after a project has been indexed. Stratus, for its part, can then query the indexed status of the project via the Cosmo API and compare it independently.

3.2.6.44. Stratus Project Status Action

With the help of the Stratus Project Status Action it is possible to lock or unlock a project in Stratus. This makes it possible for Stratus to follow Helmut's lock status at any time, so projects that have been opened and locked via Helmut can also be locked in Stratus at the same time.

3.2.6.45. Stratus Transfer Asset Action

With the help of the Stratus Transfer Asset Action it is possible to add an asset to Stratus by using the legacy Stratus system.

3.2.6.46. Stratus Trigger Momentum Action

With the help of the Trigger Momentum Action it is possible to trigger a workflow in Momentum for an asset with a defined ID.

3.2.6.47. Stratus Add Asset to Folder Action

With the help of the Stratus Add Asset To Folder Action it is possible to assign an asset to a folder. There is the opportunity to grab the folder's ID for the subsequent reference request.

3.2.6.48. Stratus Get All Folders Action

With the help of the Get All Folders Action it is possible to write a JSON response as a string into the last node result for debugging purpose. It can be used as an output to a file for example.

3.2.6.49. Stratus Create Folder Action

With the help of the Stratus Create Folder Action it is possible to test if the existing folder object will be returned after trying to add it once again.

3.2.6.50. Stratus Update Project Path Action

With the help of the Stratus Update Project Path Action it is possible to update a project's path for a given project ID.

3.2.6.51. Stratus Asset Add Folder Membership Action

With the help of the Stratus Asset Add Folder Membership Action it is possible to add an asset to a virtual folder in Stratus by entering the virtual folder path for a given asset ID.

3.2.6.52. Aurora Submit Job Action

With the help of the Aurora Action it is possible to access the Aurora service via API URL (e.g. `http://auroramaster:1002/ApiService`) and add QC jobs to the queue. With the help of that, Aurora can check footage relating to errors (e.g. validate absence) and the whole progress as well as the error count will be shown within Helmut's dashboard. The results can be accessed through the Aurora report page.

3.2.6.53. RevApp Share Asset By Email Action

With the help of RevApp Share Asset By Email Action it is possible to upload assets to RevApp and share them with E-Mail recipients. For this the E-Mail address for multiple people can be inserted in the field 'Receivers' in comma separated order. Several properties can be edit in the node's settings e.g. Max view count, expire date and all permissions of the receiver.

3.2.6.54. RevApp Share Asset Public Link Action

With the help of RevApp Share Asset Public Link Action it is possible to upload assets to RevApp and share them via a public link. Within this node the permissions for external users are restricted.

3.2.6.55. RevApp Share Asset with Internal Group Action

With the help of RevApp Share Asset with Internal Group Action it is possible to upload assets to RevApp and share them with internal groups. All permissions of the group/s in RevApp can be set within the node.

3.2.6.56. RevApp Upload Asset To Namespace Action

With the help of RevApp Upload Asset To Namespace Action it is possible to copy or move an asset to a certain namespace in RevApp by giving the file path. The messages which should be showed during and after the upload can be set within the node. Moreover there is the possibility to overwrite the original creator with a new desired one in RevApp.

3.2.6.57. RevApp Delete Asset From Namespace Action

With the help of RevApp Delete Asset From Namespace Action it is possible to remove an asset from a certain namespace in RevApp.

3.2.6.58. Hiscale Jobs Start Process Action

With the help of Hiscale Jobs Start Process Action it is possible to start a Hiscale process for a given file by referring to the corresponding URL. For that you need to authorize with username and password within the node's setting. Also the name of the target workflow as well as the file name the process is started for can be defined. On top of that several properties e.g. the desired tag, process variables and extract values from the process variables to create the node result can be set. The integration will be support for job version v2.6.0.

3.2.6.59. VPMS Check-in Action

With the help of VPMS Check-in Action it is possible to integrate VPMS (Video Production Management Suite) into Helmut4. For that an check-in will be executed via the VPMS check-in webservice and a Video or Audio file (e.g. Codec MFXOp1a HDF02a) can be transferred from share location. To use this interface the VPMS connector endpoint should be implemented in the portainer's stack first. The URL of this endpoint must be inserted in the node's settings as well as the path of the file. Furthermore the user can decide if the corresponding file should be a growing file.

The endpoint in the portainer's stack will look as following:

```
version: „3.4“
services:
  vpms-container:
    image: repo.moovit24.de:443/vpms-connector:0.0.10
    environment:
      parameters:
        --vpms.service.address=http://sp-fbpkons-vfu01.rbb-online.de:10160/CheckInService
        --vpms.idprefix=vpms://RBBFBPkons/
        --server.port=8085
    ports:
      - 8080:8080
```

3.2.6.60. VPMS Metadata Update Action

With the help of VPMS Metadata Update Action it is possible to update a video file in VPMS. According to the previous VPMS Check-in Action node the node refers to the corresponding endpoint URL. By giving the file path and the object ID of the desired VPMS object, the metadata of the video file will be updated.

3.2.6.61. Elements Add Permissions To Workspace Action

With the help of Elements Add Permissions To Workspace Action it is possible to add permissions for one or more users or groups to a certain workspace in Elements. It is mandatory to provide the Elements server URL as well as the particular user based API token. Furthermore the switch „Read only“ can limit the permissions to read-only. The node result will be the corresponding workspace name.

3.2.6.62. Elements Add Users To Groups Action

With the help of Elements Add Users To Groups Action it is possible to add multiple users to one or more groups. It is mandatory to provide the Elements server URL as well as the particular user based API token. All given „Username(s)“ will be added to every single group defined in the field „Group name(s)“.

3.2.6.63. Elements Compare To Helmut User Action

With the help of Elements Compare To Helmut User Action it is possible to compare the Elements user (provided by the API token) to the currently logged in Helmut user. It is mandatory to provide the Elements server URL as well as the particular user based API token. The node will succeed if the Elements username is matching the Helmut username. The node result will be the Elements username.

3.2.6.64. Elements Copy Files Action

With the help of Elements Copy Files User Action it is possible to execute a copy process on the server side. It is mandatory to provide the Elements server URL as well as the particular user based API token. One or more files will be copied from the source file path(s) to the provided destination directory path.

Example:

Source file path(s): /data/snfs1/.projects/media/movie.mxf

Destination directory path: /data/snfs1/.projects/approval

3.2.6.65. Elements Create API token Action

With the help of Elements Create API token Action it is possible to create a new API token on the active user. The new generated token can be named and the token itself can be used by any other user. It is mandatory to provide the Elements server URL as well as the particular user based API token. The node result will be the API Token.

3.2.6.66. Elements Create Groups Action

With the help of Elements Create Groups Action it is possible to create one or more new groups in Elements. It is mandatory to provide the Elements server URL as well as the particular user based API token. The name for the new created group(s) can be set in the field „Group name(s)“. Since Elements and Helmut allow different character sets and in order to avoid any conflicts we recommend using only alphanumeric characters, lower cases and underscores.

Example of non-recommended and recommended Group names:

- not group_100%, but group_100
- not group_A, but group_a
- not admin-group, but admin_group

3.2.6.67. Elements Create Directories Action

With the help of Elements Create Directories Action it is possible to create one or more new directories inside the Elements filesystem. It is mandatory to provide the Elements server URL as well as the particular user based API token. By entering the desired name for the new directory, the name of the media root directory under which the directory will be created should be provided in the field „Media root name“. The node result will be the media root path.

3.2.6.68. Elements Create Media Root Action

With the help of Elements Create Media Root Action it is possible to create a new media root. It is mandatory to provide the Elements server URL as well as the particular user based API token. Beside the desired „Media root name“ and the corresponding „Media root path“ for the new media root it is necessary to provide the „Volume name“ of the to be stored volume as well. The (display) name can be compared in the Status menu of Elements. The node result will be the media root name.

Example:

Media root name: media

Media root path: .projects/company

Volume name: snfs

3.2.6.69. Elements Create Production Action

With the help of Elements Create Production Action it is possible to create a new production. It is mandatory to provide the Elements server URL as well as the particular user based API token. The „Production name“ must be set in the corresponding field. According to Elements the new production can be based on a template by entering optionally the desired „Template name“ of an already existing production. If the switch „Copy content from template“ is enabled all files and folders will be copied from the given template to the new created production. Moreover the production and all included workspaces can be assigned to a certain „Default group“ and therefore be only accessible for a defined user group. The node result will be the production name.

3.2.6.70. Elements Create Share Action

With the help of Elements Create Share Action it is possible to create a new share. It is mandatory to provide the Elements server URL as well as the particular user based API token. The „Share name“ as well as the „Volume name“ must be entered. The (display) name of the volume can be extracted from the Status menu of Elements. Optionally the „Path“ of the new share can be set. Regarding the related server communication of the file sharing different protocols can be enabled or disabled. The options are SMB, NFS and AFP. Furthermore, it can be defined which groups should be able to have write access and which one should have read-only access. Certain groups can be entered in the corresponding fields. The node result will be the share name.

3.2.6.71. Elements Create Users Action

With the help of Elements Create Users Action it is possible to create one or more new users in Elements. It is mandatory to provide the Elements server URL as well as the particular user based API token. „New user account(s)“ can be created by entering key/value-pairs consisting of username and password. Since Elements and Helmut allow different character sets, to avoid any conflicts we recommend using only alphanumeric characters and underscores for the usernames. When enabling the switch „Require password change“ the users will be forced to change the password after the first login. For that the users must know the current password defined in the node settings.

Example of non-recommended and recommended User names:

- not user_100%, but user_100
- not Dan White, but dan_white
- not admin-user, but admin_user

3.2.6.72. Elements Create Workspace Action

With the help of Elements Create Workspace Action it is possible to create a new workspace. It is mandatory to provide the Elements server URL as well as the particular user based API token. The entered „Workspace name“ will be added to the desired production which can be set in the field „Production name“. The (display) name of the volume where the new workspace should be created on must be provided in the field „Volume name“. It can be compared in the Status menu of Elements. By providing a valid „Template name“ the related workspace will be used as template for the new workspace. Optionally the switch „Avid bin locking“ can be activated to emulate an Avid Nexis share for the new workspace. In order to set the hard quota for that workspace, the field „Quota size“ allows to insert a certain value in GB. The node result will be the workspace name.

3.2.6.73. Elements Delete Files Action

With the help of Elements Delete Files Action it is possible to delete one or more files from the server side. It is mandatory to provide the Elements server URL as well as the particular user based API token. The files that are supposed to be deleted can be defined by inserting the „File path(s)“. There will be no node result delivered.

Note: All file paths in the array must be valid otherwise the node will fail due to permission conflicts.

3.2.6.74. Elements Delete Files From Media Library Action

With the help of Elements Delete Files From Media Library Action it is possible to delete one or more files from a media library. It is mandatory to provide the Elements server URL as well as the particular user based API token. The reference of the deletion are the „File name(s)“ that are existing in a particular „Media library name“. There will be no node result delivered.

Note: All file paths in the array must be valid otherwise the node will fail due to permission conflicts.

3.2.6.75. Elements Delete Groups Action

With the help of Elements Delete Groups Action it is possible to delete one or more groups in Elements. It is mandatory to provide the Elements server URL as well as the particular user based API token. All the groups that need to be deleted must be entered as a comma separated list „Group name(s)“. There will be no node result delivered.

3.2.6.76. Elements Delete Directories Action

With the help of Elements Delete Directories Action it is possible to delete one or more directories from a media root by providing the to be deleted „Directory name(s)“. It is mandatory to provide the Elements server URL as well as the particular user based API token. The node result will be the media root path.

3.2.6.77. Elements Delete Media Root Action

With the help of Elements Delete Media Root Action it is possible to delete a media root and the whole content including directories and files by providing the corresponding „Media root name“. It is mandatory to provide the Elements server URL as well as the particular user based API token. There will be no node result delivered.

3.2.6.78. Elements Delete Production Action

With the help of Elements Delete Production Action it is possible to delete a production by providing the corresponding „Production name“. It is mandatory to provide the Elements server URL as well as the particular user based API token. There will be no node result delivered.

Note: Make sure the Elements API Token has the required permissions.

3.2.6.79. Elements Delete Share Action

With the help of Elements Delete Share Action it is possible to delete a share by providing the corresponding „Share name“. It is mandatory to provide the Elements server URL as well as the particular user based API token. There will be no node result delivered.

Note: Make sure the Elements API Token has the required permissions.

3.2.6.80. Elements Delete Users Action

With the help of Elements Delete Users Action it is possible to delete one or more users by providing the corresponding „User name(s)“. It is mandatory to provide the Elements server URL as well as the particular user based API token. There will be no node result delivered.

Note: Make sure the Elements API Token has the required permissions.

3.2.6.81. Elements Delete Workspaces Action

With the help of Elements Delete Users Action it is possible to delete one or more workspaces from a production by providing the corresponding „Workspace name(s)“. It is mandatory to provide the Elements server URL as well as the particular user based API token. There will be no node result delivered.

Note: Make sure the Elements API Token has the required permissions.

3.2.6.82. Elements Get Client Key Action

With the help of Elements Get Client Key Action it is possible to validate the existing client key or return a new one. The request must be manually accepted in the client itself. If the value for the „Client key“ is invalid or empty a new one will be returned. The node result will be the authorization client key .

3.2.6.83. Elements Get Free Space Of Workspace Action

With the help of Elements Get Free Space Of Workspace Action it is possible to get the remaining free space of a certain workspace by defining the „Workspace name“. It is mandatory to provide the Elements server URL as well as the particular user based API token. The node result will be the free space of the workspace.

3.2.6.84. Elements HTTP Request Action

With the help of Elements HTTP Request Action it is possible to execute HTTP requests. It is mandatory to provide the Elements server URL as well as the particular user based API token. The node can be used in the same way as the [HTTP Request Action](#) node. The node result will be the server response.

3.2.6.85. Elements Mount Workspaces On Client Action

With the help of Elements Mount Workspaces On Client Action it is possible to mount specified workspaces into the local Elements client. It is mandatory to provide the Elements server URL as well as the particular user based API token. The workspaces that are supposed to be mounted can be inserted in the array field „Workspace name(s)“. By providing the „Client key“, which can be grabbed using the [Elements Get Client Key Action](#) node, the user is able to authorize against the local client. Otherwise it is required to manually confirm the execution by clicking „Allow“ inside the Elements client. It is also possible to restrict permissions to „Read only“ by enabling the corresponding switch within the node settings. There will be no node result delivered.

3.2.6.86. Elements Move Files Action

With the help of Elements Move Files Action it is possible to execute a move operation of files from the server side. It is mandatory to provide the Elements server URL as well as the particular user based API token. By providing one or more valid „Source file path(s)“ the corresponding files will be moved to the given „Destination directory path“. The node result will be the destination path.

3.2.6.87. Elements Remove Permissions From Workspace Action

With the help of Elements Remove Permissions From Workspace Action it is possible to remove permissions from users and groups from a certain workspace. It is mandatory to provide the Elements server URL as well as the particular user based API token. By providing the related „Workspace name“ the permissions for one or more defined „Username(s)“ or „Group name(s)“ will be permanently removed. The node result will be the workspace name.

3.2.6.88. Elements Remove Users From Groups Action

With the help of Elements Remove Users From Groups Action it is possible to remove one or more users from a certain group by filling out the corresponding fields. It is mandatory to provide the Elements server URL as well as the particular user based API token. To save the files from the removed user/s after the access to a space will get lost the files can be saved to trash by enabling the corresponding switch. There will be no node result delivered.

3.2.6.89. Elements Start Job Action

With the help of Elements Start Job Action it is possible to operate jobs within the „Automation“ tab of Elements. The node will start the Elements job and query the task for progress. It is mandatory to provide the Elements server URL as well as the particular user based API token. The provided „Job name“ is the name of the job that is going to be executed. All types of jobs that are available in Elements will be supported. Optionally one or more „Job variables“ (key+value) can be attached to the corresponding job. The node result will be the new task ID.

Note: It is important that the given „Job name“ is a unique entry in the database of Elements. In case multiple job name entries with the same name exist the node will fail. Possibly the assistance of an Elements Admin is needful to ensure that prerequisite.

3.2.6.90. Elements Unmount Workspaces From Client Action

With the help of Elements Unmount Workspaces From Client Action is possible to unmount one or more workspaces from the local Elements client by providing the „Workspace name(s)“. It is mandatory to provide the Elements server URL as well as the particular user based API token. If the „Client Key“ which can be grabbed using the [Elements Get Client Key Action](#) node will not be inserted, you need to manually allow this action in the Elements client. There will be no node result delivered.

3.2.6.91. Elements Update Workspace Quota Action

With the help of Elements Update Workspace Quota Action it is possible to update the hard quota of a certain workspace by providing the „Workspace name(s)“. It is mandatory to provide the Elements server URL as well as the particular user based API token. The „Quota size“ in GB must be entered as well and only supports integer values in the corresponding field. The node result will be the new generated quota.

3.2.6.92. Elements Upload File Into Media Library Action

With the help of Elements Upload File Into Media Library Action it is possible to upload a certain file into a desired media library by providing the „Source file path“ of that file including the name and the extension. Additionally the particular „Media library name“ must be entered in the related field. It is mandatory to provide the Elements server URL as well as the particular user based API token. The node result will be the new source file path.

3.2.7. User

3.2.7.1. Helmut Add Users To Group Action

With the help of the Helmut Add Users To Group Action it is possible to add one or multiple users to a group. This can be used, for example, within a create_user stream to add those users to a specific group.

3.2.7.2. Helmut Remove Users From Group Action

With the help of the Helmut Remove Users From Group Action it is possible to remove one or multiple users from a group. This can be used analogue to the opposite node Helmut Add Users To Group Action.

3.2.7.3. Helmut Input Dialog Action

With the help of the Helmut Input Dialog Action, it is possible to send a message to a user on the web with the request to enter a value. This value can be used as the node result in the running stream. The user to whom this notification is sent must be online and logged in to receive the message. If the user is not signed in, the stream will fail. Furthermore, it is possible to specify in the node whether the node should fail or be successful if the user does not make the entry. The time available to the user to make the entry can also be defined.

3.2.7.4. Helmut Confirm Dialog Action

With the help of the Helmut Confirm Dialog Action, it is possible to send a confirm dialog to a user on the web. The user to whom this dialog is sent must be online and logged in to receive the message. If the user is not signed in, the stream will fail. Furthermore, it is possible to specify in the node whether the node should fail or be successful if the user does not make the entry. The time available to the user to make the entry can also be defined.

3.2.8. File and Folder Actions

3.2.8.1. File Copy Action

With the help of the File Copy Action it is possible to copy a file from one path to another. This node is mainly intended to copy individual files such as templates or preference files. To copy video files it is recommended to use the "Job File Copy Action" in an asynchronous way of a stream, as this can show the progress in the dashboard. (requires HIO)

3.2.8.2. File Create Action

With the help of the File Create Action it is possible to create one or more files in defined paths. These can be files that can be generated at the operating system level (stream executing OS). This node could be used, for example, to create a .txt file into which data from the stream is written.

3.2.8.3. File Delete Action

With the help of the File Delete Action it is possible to delete one or more files.

3.2.8.4. File Increment Name Action

With the help of the File Increment Action it is possible to increase the name of a file by adding a number. This works in such a way that the file path is specified and the result of the node is an increased file name (including path). This increased filename can in turn be used via the wildcard {stream.last_result} in the next node in the stream or via the wildcard {node.result.?.} anywhere in the stream. This node is helpful when you neither want to overwrite an existing file nor skip the process for the current file to be processed.

3.2.8.5. File Move Action

With the help of the File Move Action it is possible to move a file from "a" to "b". In contrast to the File Copy Action, the file is removed from the source path. This node is mainly intended to move individual files such as templates or preference files. To move video files it is recommended to use the "Job File Copy Action" in an asynchronous way of a stream, as this can show the progress in the dashboard. (requires HIO)

3.2.8.6. File Open Action

With the help of the File Move Open Action it is possible to open a file. The file is always opened with the program associated with this type of file on the operating system. This makes it possible to use this node for a large number of programs. The main use case is opening the project files (Premiere, After Effects, Audition).

The node is just triggering the OS and the operating system is the parent process under which the program assigned to the file to be opened runs.

3.2.8.7. File Rename Action

With the help of the File Rename Action it is possible to rename a file, for example to add a prefix or appendix to a project file after it has been duplicated.

3.2.8.8. File Replace Content Action

With the help of the File Replace Content Action it is possible to replace the content of a file that can be read, opened and changed by the operating system. To do this, it is necessary to know the content to be replaced, since this must be specified in the form of a string in the node. It is possible to replace the defined string with any value available in the stream.

The node can be used, for example, to replace the name of a sequence or the paths of the existing assets within the project.

3.2.8.9. Folder Content Delete Action

With the help of the Folder Content Delete Action it is possible to delete the content of one or more folders. The content is deleted recursively. The specified folder itself is retained.

3.2.8.10. Folder Copy Action

With the help of the Folder Copy Action it is possible to copy a folder including its contents from "a" to "b". This node is mainly intended to copy folders with a small size. To copy folders that contain video assets, it is recommended to use the Job Folder Copy Action in an asynchronous way of the stream, as this allows the progress of the copy process to be displayed in the dashboard. (requires HIO)

3.2.8.11. Folder Create Action

With the help of the Folder Create Action it is possible to create one or more folders in a defined path. The number of folders is not limited.

3.2.8.12. Folder Delete Action

With the help of the Folder Delete Action it is possible to delete one or more folders. The entire folder including its contents is deleted.

3.2.8.13. Folder Increment Name Action

With the help of the Folder Increment Action it is possible to increase the name of a folder by adding a number. This works in such a way that the folder path is specified and the result of the node is an increased folder name (including path). This increased folder name can in turn be used via the wildcard {stream.last_result} in the next node in the stream or via the wildcard {node.result.?.} anywhere in the stream. This node is helpful when you neither want to overwrite an existing folder nor skip the process for the current folder to be processed.

3.2.8.14. Folder Move Action

With the help of the Folder Move Action it is possible to move a folder from "a" to "b". In contrast to the Folder Copy Action, the folder is removed from the source path. This node is mainly intended to move folders with a small size. To move folders that contain video assets, it is recommended to use the Job Folder Copy Action in an asynchronous way of the stream and then delete the original folder, as this allows the progress of the copy process to be displayed in the dashboard. (requires HIO)

3.2.8.15. Folder Rename Action

With the help of the Folder Rename Action it is possible to rename a folder, for example to add a prefix or appendix.

3.2.8.16. XSquare File Check In Action

With the help of the XSquare File Check In Action it is possible to transfer an asset to XSquare and assign it to a target there. This makes it possible from Helmut to ensure that the transferred file is processed further within XSquare, for example transferred to IP Director.

3.2.8.17. 7-Zip Archive Action

With the help of the 7-Zip Archive Action it is possible to add files and folders at a desired location to a new zip archive. A previous installation of 7-Zip is required to use this node.

3.2.8.18. 7-Zip Extract Action

With the help of the 7-Zip Extract Action it is possible to extract the content of a certain file archive to the current level or to a desired location. A previous installation of 7-Zip is required to use this node.

3.2.8.19. 7-Zip CLI Action

With the help of the 7-Zip CLI Action it is possible to execute common „add“ and „xtract“ commands. This is the advanced mode to use archiving as well as extracting procedures. This nodes provides the following switches and further appendices: ao, ax, i, m, o, p, r, scs, t, v, w, x, y and ssw.

3.2.8.20. Get Files From Folder Action

With the help of the Get Files From Folder it is possible to iterate over file paths of a folder. The files for a desired folder path can be scanned only on top level or in recursive way by enabling corresponding switch within the node settings. When proceeding with the node's output „Next file“ all found file paths will be written into the job.source wildcard iteratively.

3.2.8.21. Get Folder From Folder Action

With the help of the Get Folders From Folder it is possible to iterate over folder paths of a folder. The folder for a desired main folder path can be scanned only on top level or in recursive way by enabling corresponding switch within the node settings. When proceeding with the node's output „Next folder“ all found folder paths will be written into the job.source wildcard iteratively.

3.2.9. OS Action Nodes

3.2.9.1. Commandline Execute Action

With the help of the Commandline Execute Action it is possible to invoke a shell command on either the client or server. The Commandline Execute Action supports the windows, Mac, and unix shells. This makes it possible to trigger external third-party tools from a stream.

3.2.9.2. Unmount A Share Action

With the help of the Unmount A Share Action it is possible to unmount a share that has been mounted via smb.

3.2.10. MISC Action Nodes

3.2.10.1. Execute Javascript Action

With the help of the Execute Javascript Action it is possible to execute Javascript code based on NASHORN. The resulting possibilities are diverse. The main focus is on the integration of third-party systems that MoovIT GmbH does not yet have in the form of one or more standardized action nodes. If you need any help, please contact us.

3.2.10.2. Fail Action

With the help of the fail action it is possible to ensure that a node or an entire stream fails even though it is successful. Why would anyone want to do this?

It is conceivable that the behavior of a node is applied differently than the node per se implies. For example, a condition that asks something and if that is true, you want to ensure that the stream is not successful. This node is the solution.

3.2.10.3. HTTP Request Action

With the help of the HTTP request action it is possible to execute HTTP requests and thus implement integrations between Helmut and third-party systems via a stream. The available methods are Get, Post, Put and Delete. The response code for the request will be provided in the node's response object in the field "code" and can be retrieved, altogether, by using the wildcard {node.result.?}.

3.2.10.4. JSON Extract Action

With the help of the Json Extractor Action it is possible to access the key value from a Json payload. This action can be used, as an example, together with the FFProbe as Json Action and the MedialInfo as Json Action to retrieve the key value from one of these actions. The Node is used like this:

If the payload looks like this:

```
{
  "id": 1,
  "sequenceID": "cba14e1e-fef8-4624-9679-b8d087140345",
  "name": "Breadcrumb Demo 21-04-2021 09-58-44 david",
  "audioTracks": {
    "numTracks": 8
  },
  "videoTracks": {
    "numTracks": 3
  },
  "frameSizeHorizontal": 1920,
  "frameSizeVertical": 1080,
```

```
"timebase": "10160640000",
"zeroPoint": "0",
"end": "4897428480000",
"markers": {
  "numMarkers": 0
}
```

The complete payload has to be added to the Node (see figure 3.2.2) If the frame size should be extracted you need to add "frameSizeHorizontal" as Key path (see figure 3.2.2). The result will be: 1920.

If you only know the path to a .JSON file you need to read out the content of that file and use the content as the input for the Node.

Example:

- If the file is called: **/Volumes/myfiles/myfile.json**
- And the content of the file looks like this:

```
{
  "id": 1,
  "sequenceID": "cba14e1e-fef8-4624-9679-b8d087140345",
  "name": "Breadcrumb Demo 21-04-2021 09-58-44 david",
  "audioTracks": {
    "numTracks": 8
  },
  "videoTracks": {
    "numTracks": 3
  },
  "frameSizeHorizontal": 1920,
  "frameSizeVertical": 1080,
  "timebase": "10160640000",
  "zeroPoint": "0",
  "end": "4897428480000",
  "markers": {
    "numMarkers": 0
  }
}
```

- Add this as the Payload: **{file.content./Volumes/myfiles/myfile.json}**



Figure 3.2.2 JSON Extract

3.2.10.5. Metadata Auto Mapper Action

With the help of the Metadata Auto Mapper Action it is possible to resolve wildcards, which were defined in metadata as default value, in the stream, since otherwise they would appear in the pure form as {wildcard} in the metadata of the respective object.

A suitable use case would be the use of a timestamp in a hidden metadata field.

3.2.10.6. Regex Apply Action

With the help of the Regex Apply Action it is possible to clean a string (for example a project name) via a REGEX. For example, if the permitted characters were restricted to a-A and 0-1 and this REGEX was applied to the string, all characters that do not meet this requirement would be replaced. It is possible to define with which other characters these should be replaced. The conformed string is available via the wildcard {stream.last_result} or via {node.result.?.} And can thus be used in the stream in the corrected form.

3.2.10.7. Sleep Action

With the help of the sleep action it is possible to ensure that the stream does not continue for the period specified in the sleep node. Why should someone do this?

For example, it is conceivable that a node sends an API call to a third party system (Javascript action) and the corresponding API does not return a response. Accordingly, the node cannot wait for the answer and would be successful in terminating the call. If you now want to do something with the information from the third party system in the following node and make another call, it could mean that the first one has not yet been completed. In this case the sleep node would help to artificially lengthen the stream between the two nodes.

3.2.10.8. Split Stream Action

With the help of the Split Stream Action, a stream can be split into a synchronous and an asynchronous path. The synchronous path of a stream must always end within 60 seconds, i.e. all nodes that are in the synchronous path must have been processed. For example, this cannot actually be guaranteed for copying and rendering processes. Such processes should be in an asynchronous way when more than one node is involved.

The split stream node enables this asynchronous path, which is inherently synchronous, to start at any point in the stream.

3.2.10.9. Stream Execute Generic Stream Action

With the help of the Stream Execute Generic Stream Action, it is possible to execute an existing stream using its ID. This node represents the counterpart to the Job Create Job Action, via which a JOB can be created using an existing profile ID. Since FX streams are not controlled via profiles, but are usually executed directly, they can also be triggered within a stream via the present node.

3.2.10.10. Stream Set Temporary Variable Action

With the help of the Stream Set Temporary Variable Action it is possible to set a variable in the stream and to query it at any point in the stream using the wildcard {stream.variable.?.}. For example, a path that occurs repeatedly within the stream would be easier to retrieve. The variable is no longer available after the stream has ended, but it is available in a directly following stream, for example if it was set in a pre-stream. It will be set again the next time the same stream is executed.

3.2.10.11.Stream Set Store Variable Action

With the help of the Stream Set Store Variable Action it is possible to set a variable in the Variable Store within a stream. It is possible to define the name (key) and the value of the variable. As soon as a variable has been set, it is available in the stream itself and globally for all other streams that follow in time.

3.2.10.12.Stream Get Store Variable Action

With the help of the Stream Get Store Variable Action, it is possible to read out a variable that exists in the Variable Store by specifying the name (key) of the variable. The value of the variable is then available in the form of the node results and can be read out using the wildcard `{node.result.?`.

3.2.10.13.Stream Delete Store Variable Action

With the help of the Stream Delete Store Variable Action it is possible to delete a variable existing in the Variable Store by specifying the name (key) of the variable. This is only possible if the variable is not protected. Variables can be protected in the variable store. See: *WIP - H4 - Streamdesigner Guide - en.pdf*

3.2.10.14.Success Action

With the help of the success action it is possible to ensure that a node or an entire stream is successful even though it fails. Why should someone do this?

It would be conceivable, for example, that a stream triggered by the "create_user" event should copy something. The user is only created if the stream is successful. If the copying process is unsuccessful for any reason, no user will be created. Since the "create_user" event is also triggered if, for example, a user is added via the AD (and that can be many at once), no user would be created. In this case, the fail output of the copy node could be linked to a success action to ensure that the stream is always successful. In doing so, however, you have to accept that the copying process may not be carried out and you would not notice this.

3.2.10.15.XPath Action

With the help of the XPath Action it is possible to extract specific XPath expression from XML file. By returning an extracted XML value from the node the expression can be placed into the metadata of a job. Using this node realizes the stable communication with third parties like TIXstream.

3.2.10.16.XML Generator Action

Coming Soon!

3.3. Output Nodes

3.3.1. File and Folder Output

3.3.1.1. Write File Output

With the help of the Write File Output Action it is possible to write all or part of the payload to a file at any point within a stream. Any file format that can be written by the operating system can be used for this. Example:

If you want to write the stream log in a text file, you enter the path including file name and file extension within the node and use the wildcard: {stream.log} as input parameters.

```
/Volumes/folder/folder/mytextfile.txt
```

3.3.2. OS Output

3.3.2.1. MacOS System Notification Output

With the help of the MacOS System Notification Action, it is possible to send a notification to a user via the operating system. In this case it is only possible for MAC and within a client stream, as the server cannot communicate with the respective workstation.

3.3.3. MISC

3.3.3.1. Send Email Output

With the help of the Send Email Output Action it is possible to send an email to one or more recipients. It is necessary to set up a mail server whose access data must be entered in the node. It is also possible to freely define the title and text of the email.

3.3.3.2. Telegram Output

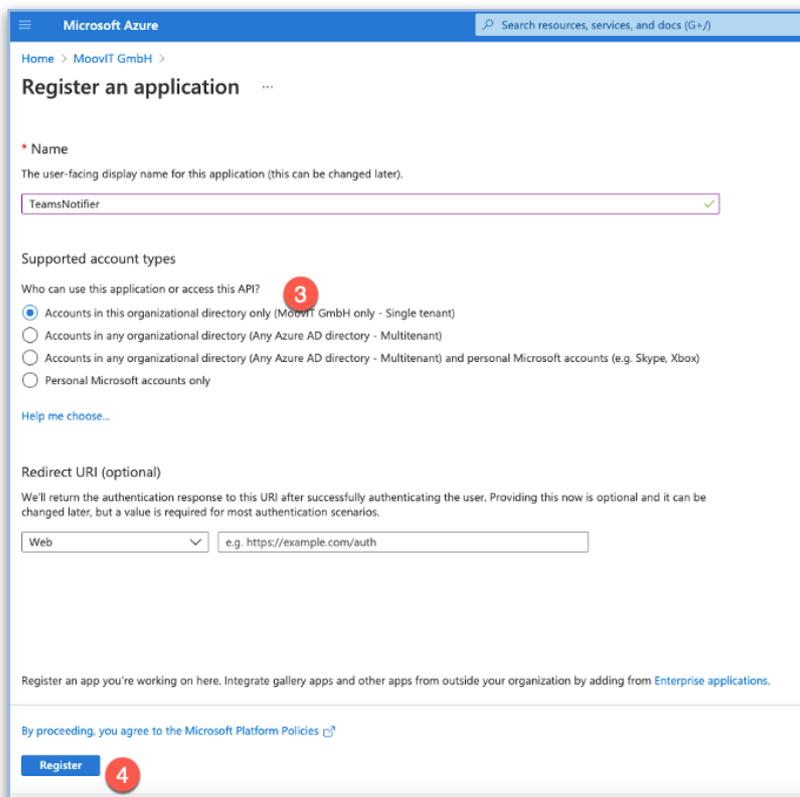
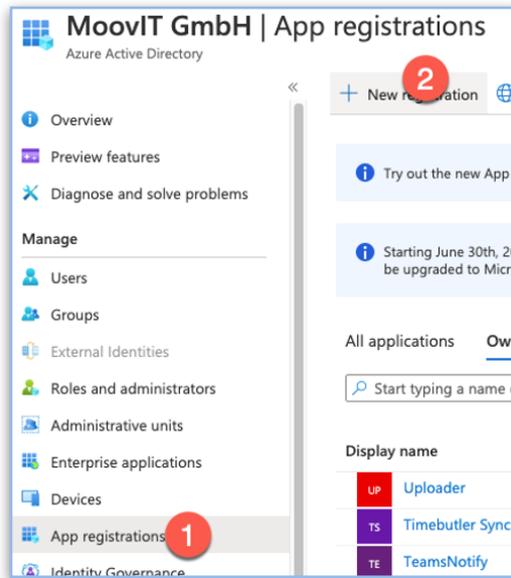
With the help of the Telegram Output Action it is possible to send a notification to a telegram group. For this you need an API token as well as the recipient ID. This can be obtained as follows:

- API key:
 - Sign up for Telegram using any application.
 - Log in to your Telegram core: <https://my.telegram.org>.
 - Go to "API development tools" and fill out the form.
 - You will get basic addresses as well as the api_id and api_hash parameters required for user authorisation.
 - For the moment each number can only have one api_id connected to it.
- Channel ID:
 - Login to your account at web version of Telegram: <https://web.telegram.org>
 - Find your channel. Browse your url, it should be like https://web.telegram.org/#/im?p=c1055587116_11052224402541910257
 - Grab "1055587116" from it, and add "-100" as a prefix.

3.3.3.3. Send Message To Teams Channel

With the help of Send Message To Teams Channel Output it is possible to send a text to a desired channel in Microsoft Teams by inserting the link of the corresponding channel. To use this feature the App registration for Azure Active Directory needs to be done first and Teams Notifier must be activated. Please follow these steps:

- Visit: https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps



Microsoft Azure | Search resources, services, and docs (G+)

Home > MoovIT GmbH > TeamsNotifier

TeamsNotifier | API permissions

Search (Cmd+/) | Refresh | Got feedback?

Configure permissions

The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value in your organization, or in organizations where this app will be used. [Learn more](#)

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list shows all the permissions the application needs. [Learn more about permissions and consent](#)

+ Add a permission | Grant admin consent for MoovIT GmbH

API / Permissions name	Type	Description	Admin consent required
Microsoft Graph (1)			
User.Read	Delegated	Sign in and read user profile	No

To view and manage permissions and user consent, try [Enterprise applications](#).

Request API permissions

Select an API

Microsoft APIs | APIs my organization uses

Commonly used Microsoft APIs

- Microsoft Graph**
Take advantage of the tremendous power of the Microsoft Graph API. Access Azure AD, Intune, Office 365, and more from a single endpoint.
- Azure Rights Management Services**
Allow validated users to read and write protected content.
- Data Export Service for Microsoft Dynamics 365**
Export data from Microsoft Dynamics CRM organization to an external destination.
- Flow Service**
Embed flow templates and manage flows.

Request API permissions

< All APIs

Microsoft Graph
https://graph.microsoft.com/ Docs

What type of permissions does your application require?

- Delegated permissions**
Your application needs to access the API as the signed-in user.
- Application permissions**
Your application runs as a background service or daemon without a signed-in user.

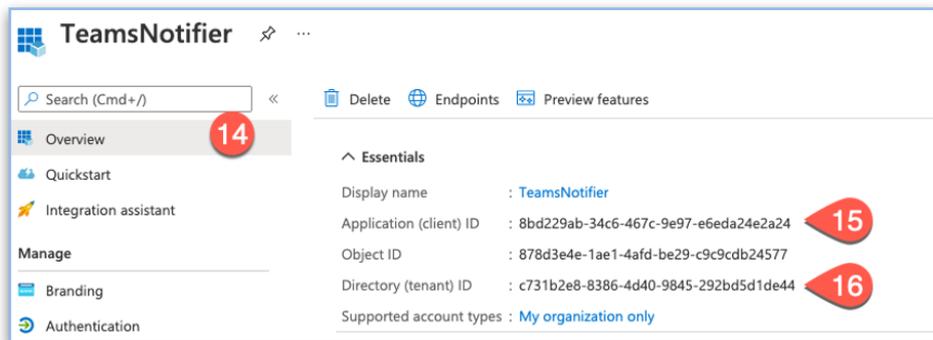
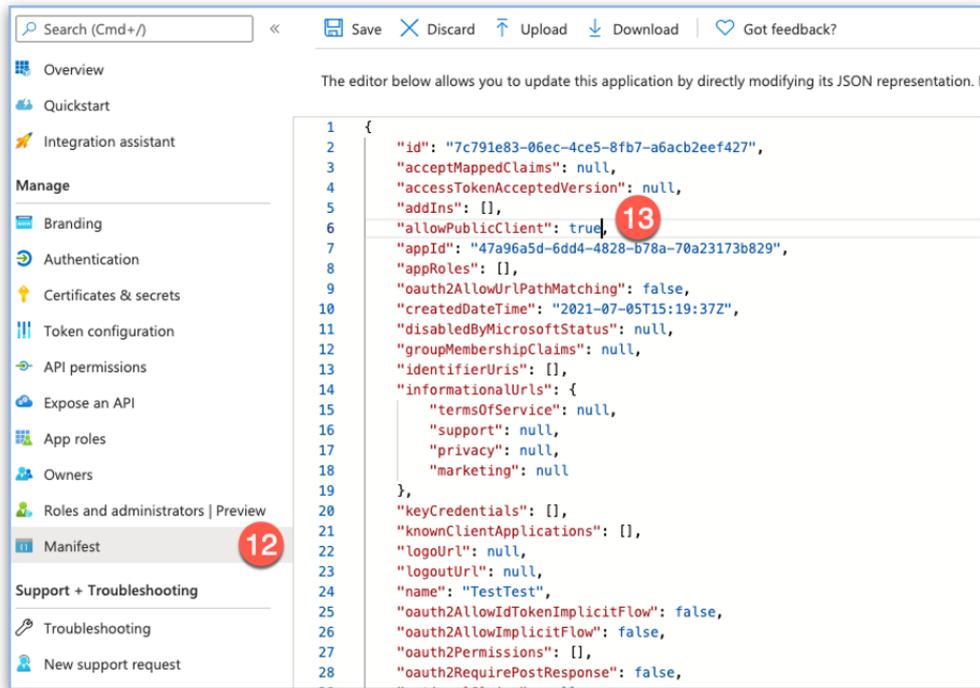
Select permissions [expand all](#)

ChannelMessage.Send

The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value in your organization, or in organizations where this app will be used. [Learn more](#)

Permission	Admin consent required
ChannelMessage (1)	
<input checked="" type="checkbox"/> ChannelMessage.Send Send channel messages	No

Add permissions | Discard



- Also the Service needs to be added to Docker with the Teams Notifier connector endpoint. The portainer's stack file must be extended with the following properties:

teams_connector:

image: repo.moovit24.de:443/teams-connector:0.9.4

restart: always

environment:

parameters:

--de.moovit.teamsnotify.graph.security.clientId=<ID Step 15>

--de.moovit.teamsnotify.graph.security.authority=https://login.microsoftonline.com/<Tenant ID Step

16/

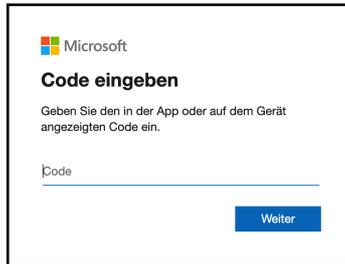
ports:

- 8471:8471

- Access the Docker IP using Port 8471 to see Swagger Doc
- Authenticate Container: <http://{dockerhost}:8471/v1/authentication/message>
- Click on the link below to sign in and enter corresponding code to authenticate

← → ↻ 🔒 Nicht sicher | 192.168.122.82:8471/v1/authentication/message

To sign in, use a web browser to open the page <https://microsoft.com/devicelogin> and enter the code CG9JCR2UG to authenticate.



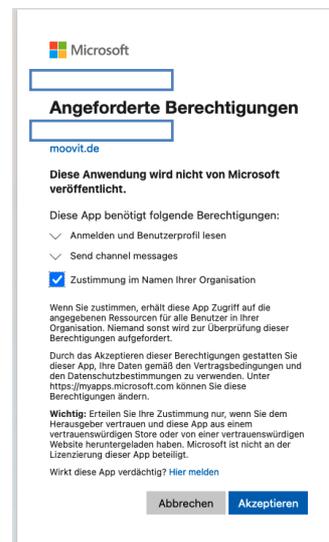
17

18

- Log in to your organization's Microsoft account and accept the permissions

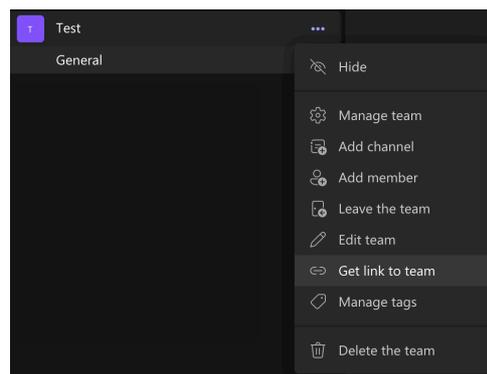


19



20

- Retrieve Channel ID and Team ID



21

- Example:

[https://teams.microsoft.com/l/team/
19%3aHDloAoPDwl0ejNcq4sjmQJHDjEERSlamAbF7dgo6w1%40thread.tacv2/conversations?
groupId=a6eb9ab3-547b-4acfbec2e59405bc9b70&tenantId=c731b2e8-8386-4d40-9845-292bd5d1de44](https://teams.microsoft.com/l/team/19%3aHDloAoPDwl0ejNcq4sjmQJHDjEERSlamAbF7dgo6w1%40thread.tacv2/conversations?groupId=a6eb9ab3-547b-4acfbec2e59405bc9b70&tenantId=c731b2e8-8386-4d40-9845-292bd5d1de44)

Team ID

Channel ID

3.3.4. User

3.3.4.1. Send Message to User

With the help of the Send Message to User Action it is possible to send a message to a user. The message appears in the FX web interface and must be confirmed by clicking OK. This can be used, for example, to inform the user about the progress of a workflow if this workflow does not appear in the dashboard (all FX streams).

3.3.4.2. Send Notification to User

With the help of the Send Notification to User Action it is possible to send a Notification to a user that appears in the FX web interface and disappears after 10 seconds. This can be used, for example, to inform the user about errors that occur or if the user is not allowed to run a specific workflow.

4. Wildcards

4.1. General Wildcards

The description for each general wildcard is stored in the Stream Designer and is displayed in the node palette. Most are self-explanatory. A few that require further explanation are listed here:

4.1.1. Job Breadcrumb Wildcard

This wildcard is replaced by the folder that is present in the job object or that is transferred to the job object. This value is set for the following actions:

- Auto import (panel required):
 - If an asset is loaded via the auto import function, the {job.breadcrumb} wildcard in the created job is replaced by the name of the folder into which the file was imported. This can also be the root folder.
- Web upload (Cosmo):
 - If an asset is loaded into Cosmo via the web upload function, the {job.breadcrumb} wildcard in the created job is replaced by the name of the folder into which the file was imported. A file must be imported into a folder within Cosmo. This can also be the root folder.

4.1.2. Job Destination Wildcard

This wildcard is always replaced by the destination path of the previous node and is always overwritten in the payload of the job object. For example, if the Job Render In AME Action is used, the Destination File Path defined there is replaced in the {job.destination} wildcard. If another action node is used after this node, which also contains a destination path, the {job.destination} wildcard is replaced with this value from then on.

4.1.3. Job Proxy Wildcard

This wildcard is always replaced by the destination path of the Job Render Proxy in AME Action if this was used in the stream.

4.2. Functional Wildcards

Functional wildcards enable interaction with the input itself by entering an input (?). A string, a file path or a wildcard itself can serve as input. It is therefore possible to nest wildcards.

4.2.1. Convert Date To Timestamp ? Wildcard

With the help of this wildcard, an existing date can be converted to a Unix timestamp (always includes date and time). The time that is set is the time at which the wildcard is used within a node. The wildcard is used in the following way:

- {convert.date.to_timestamp.>}
- For the following input (?) -> 2020-03-21
- The output is going to be -> 1584748800000 [Unix timestamp]

4.2.2. Convert Timestamp To Date ? Wildcard

With the help of this wildcard, a date, which is available in the form of a Unix timestamp, can be converted into a standardised format. The wildcard is used in the following way:

- {convert.timestamp.to_date.>}
- For the following input (?) --> 158474880000 [Unix timestamp]
- The output is going to be --> 2020-03-21

4.2.3. Convert Timestamp To Datetime ? Wildcard

With the help of this wildcard, a date which is available in the form of a Unix timestamp can be converted into a standardised format. At the same time, the current time at which this is done is added. The wildcard is used in the following way:

- {convert.timestamp.to_datetime.>}
- For the following input (?) --> 158474880000 [Unix timestamp]
- The output is going to be --> 2020-03-21 09:41:21

4.2.4. Date Day ? Wildcard

With the help of this wildcard, the specification of the day of a date can be extracted. The wildcard is used in the following way:

- {date.day.>}
- For the following input (?) --> 2020-03-01
- The output is going to be --> 01

4.2.5. Date Month ? Wildcard

The month of a date can be extracted with the help of this wildcard. The wildcard is used in the following way:

- {date.month.>}
- For the following input (?) --> 2020-03-01
- The output is going to be --> 03

4.2.6. Date Month Textual ? Wildcard

The month (in form of short text) of a date can be extracted with the help of this wildcard. The wildcard is used in the following way:

- {date.month.textual.>}
- For the following input (?) --> 2020-11-01
- The output is going to be --> November

4.2.7. Date Month Textual Short ? Wildcard

The month (in form of short text) of a date can be extracted with the help of this wildcard. The wildcard is used in the following way:

- {date.month.textual.short.>}
- For the following input (?) --> 2020-11-01
- The output is going to be --> Nov

4.2.8. Date Year ? Wildcard

This wildcard will be replaced by the year of a defined date. The wildcard is used in the following way:

- {date.year.>}
- For the following input (?) --> 2019-11-01
- The output is going to be --> 2019

4.2.9. Date Shortyear ? Wildcard

This wildcard is replaced by the year in the short form of a defined date. The wildcard is used in the following way:

- {date.shortyear.>}
- For the following input (?) --> 2019-11-01
- The output is going to be --> 19

4.2.10. File Content ? Wildcard

This wildcard is replaced by the content of a defined text-based document. The wildcard is used in the following way:

- {file.content.>}
- For the following input (?) --> c:\values.txt
- The output, for example, is going to be --> a,b,c,d

Notes:

Ensure that the render node or server has access to file. If the wildcard is used on a server-side stream, the streams container needs to have access to the file path

Streams engine needs to be able to read the file content. Use this functional wildcard to read text-based documents.

4.2.11. File Exists ? Wildcard

This wildcard is replaced by True or False, depending on whether the file that is being searched for exists or not. The wildcard is used in the following way:

- {file.exists.>}
- For the following input (?) --> R:\Users\Administrator\Pictures\helmut-IO.ico
- The output is going to be --> True or False

Notes:

Ensure that the render node or server has access to file. If the the wildcard is on a server-side stream, the streams container needs to have access to the file path.

Boolean return

4.2.12. File MD5 ? Wildcard

This wildcard is replaced by the MD5 hash of a specified file. In case of doubt, this takes a long time and the wildcard is used in the following way:

- {file.md5.>}
- For the following input (?) --> R:\Users\Administrator\Pictures\helmut-IO.ico
- The output is going to be --> a3cca2b2aa1e3b5b3b5aad99a8529074 [MD5 hash]

Notes:

Ensure that the render node or server has access to file. If the the wildcard is on a server-side stream, the streams container needs to have access to the file path.

4.2.13. File Modified ? Wildcard

This wildcard is replaced by the Unix timestamp of a specified file. In order to get to the date or the time stamp as an individual value, this wildcard must be nested with another wildcard. (see advanced example)
The wildcard is used in the following way:

- {file.modified.>}
- For the following input (?) --> R:\Users\Administrator\Pictures\helmut-IO.ico [FilePath]
- The output is going to be --> 1584748800000 [Unix timestamp]

Notes:

Ensure that the render node or server has access to file. If the wildcard is used on a server-side stream, the streams container needs to have access to the file path

Advanced example:

```
{convert.timestamp.to_datetime.{file.modified.R:\Users\Administrator\Pictures\helmut-IO.ico}}
```

4.2.14. File Size ? Wildcard

This wildcard is replaced by the file size in bytes of a specified file. The wildcard is used in the following way:

- {file.size.>}
- For the following input (?) --> R:\Users\Administrator\Pictures\helmut-IO.ico
- The output is going to be --> 256 [filesize in byte]

Notes:

Ensure that the render node or server has access to file. If the wildcard is used on a server-side stream, the streams container needs to have access to the file path

4.2.15. Folder Content ? Wildcard

This wildcard will be replaced by a comma-separated list of the files in a defined folder. The wildcard is used in the following way:

- {folder.content.>}
- For the following input (?) --> R:\Users\Administrator\Pictures
- The output is going to be --> helmut-IO.ico,helmut-CO.ico [Folder content as comma-separated values]

Notes:

Ensure that the render node or server has access to file. If the wildcard is used on a server-side stream, the streams container needs to have access to the file path

4.2.16. Folder Exists ? Wildcard

This wildcard is replaced by true or false, depending on whether the specified folder exists or not. The wildcard is used in the following way:

- {folder.exists.>}
- For the following input (?) --> R:\Users\Administrator\Pictures
- The output is going to be --> bool True/False

Notes:

Ensure that the render node or server has access to file. If the wildcard is on a server-side stream, the streams container needs to have access to the file path.

Boolean return

4.2.17. Folder Modified ? Wildcard

This wildcard is replaced by the modified date in the form of a Unix timestamp of the specified folder. The wildcard is used in the following way:

- {folder.modified.>}
- For the following input (?) --> R:\Users\Administrator\Pictures [FilePath]
- The output is going to be --> 158474880000 [Unix timestamp]

Notes:

Ensure that the render node or server has access to file. If the wildcard is on a server-side stream, the streams container needs to have access to the file path.

4.2.18. Folder Size ? Wildcard

This wildcard is replaced by the total size of all files of the specified folder. The wildcard is used in the following way:

- {folder.size.>}
- For the following input (?) --> R:\Users\Administrator\Pictures [Folder Path]
- The output is going to be --> 18521

Notes:

Ensure that the render node or server has access to the folder path . If the wildcard is on a server-side stream, the streams container needs to have access to the folder path.

4.2.19. Helmut Variable ? Wildcard

This wildcard is replaced by the value of a variable that was defined in the Helmut Preferences. The name of the variable is used as input. The wildcard is used in the following way:

- {helmut.variable.>}
- For the following input (?) --> MyVariableName
- The output is going to be --> VariableValue

4.2.20. Job metadata ? Wildcard

This wildcard is replaced by the value of a metadata entry that can be set via the metadata menu. The name of the metadata entry serves as input. The value can only be read out if there is an entry in the database for the object in the stream (job object). It is not enough for the metadata entry to exist in FX / IO.

- {job.metadata.>}
- For the following input (?) --> MyMetadataName
- The output is going to be --> MetadataValue

4.2.1. Job custom ? Wildcard

This wildcard is replaced by the custom value of a specific key set via specific nodes. It is designed to hold Asset ID from third party systems. If the Medialoopster Add Asset To Project Action is used to add an Asset to Medialoopster, the ID provided by the MediaLoopster API will be added to Helmut's database connected to the Asset ID provided by Helmut. If the wildcard is used while executing a job that is related to an Asset that holds a third party id, it will resolve the ID.

- {job.custom.>}
- For the following input (?) --> medialoopster
- The output is going to be --> medialoopster asset ID

The wildcard can be used according to the third party integrations that do support this wildcard. At the moment this is medialoopster.

4.2.1. Project custom ? Wildcard

This wildcard is replaced by the custom value of a specific key set via specific nodes. It is designed to hold project ID' from third party systems. If the „Medialoopster Create Project Action“ is used to add a project to Medialoopster, the ID provided by the MediaLoopster API will be added to Helmut's database connected to the project ID provided by Helmut. If the wildcard is used while executing a stream that is related to a project that holds a third party id, it will resolve the ID.

- {project.custom.>}
- For the following input (?) --> medialoopster
- The output is going to be --> „medialoopster project ID

The wildcard can be used according to the third party integrations that do support this wildcard. At the moment this is medialoopster.

4.2.2. Local Environment ? Wildcard

This wildcard is replaced by the value of an environment variable. This variable must be available on the OS that is executing the stream (previously set).

{local.environment.>}

For the following input (?) --> MyLocalEnvironmentVariableName

The output is going to be --> TheValue

4.2.3. Local Profile ? Wildcard

This wildcard will be replaced by the path of the local Premiere Pro profile. Since there is a different path for each major version of Adobe Premiere Pro, it is sufficient to enter the number of the version here as an input.

- {local.profile.>}
- For the following input (?) --> 14 (Premiere Major Version)
- The output is going to be --> root\documents\Adobe\Premiere Pro\14.0\Profile-root

4.2.4. Local Registry ? Wildcard

{local.registry.??}

For the following input (??) --> Registry hive

The output is going to be --> Registry key value

Example:

For the following input --> {local.registry.currentuser/SOFTWARE/Microsoft/Windows/CurrentVersion/Explorer/Shell Folders/Local AppData}

The output is going to be --> C:\Users\EDIT\AppData\Local

Notes:

Registry keys should start like this:

- classesroot
- currentuser
- localmachine
- users
- currentconfig
- Registry path are / (forward slashes) instead of the usual \ (back slash)

4.2.5. Path Basename ? Wildcard

This wildcard is replaced with the filename of a path (without extension). Any path can serve as input. The wildcard is used in the following way:

- {path.basename.??}
- For the following input (??) --> c:\Users\Administrator\Pictures\helmut-IO.ico
- The output is going to be --> helmut-IO

Notes:

Ensure that the render node or server has access to file. If the wildcard is on a server-side stream, the streams container needs to have access to the file path.

4.2.6. Path Extension ? Wildcard

This wildcard is replaced with the extension of a file path. Any path can serve as input. The wildcard is used in the following way:

- {path.extension.??}
- For the following input (??) --> c:\Users\Administrator\Pictures\helmut-IO.ico
- The output is going to be --> ico

Notes:

Ensure that the render node or server has access to file. If the wildcard is on a server-side stream, the streams container needs to have access to the file path.

4.2.7. Path Map To Unix ? Wildcard

This wildcard is replaced with the Unix syntax of a Windows path. The wildcard is used in the following way:

- {path.map.to.unix.??}
- For the following input (??) --> R:\Users\Administrator\Pictures\helmut-IO.ico
- The output is going to be --> /Volumes/Profiles_1/Users/Administrator/Pictures/helmut-IO.ico

Note:

This functional wildcard returns the value from the path mapping table.

4.2.8. Path Map To Win ? Wildcard

This wildcard is replaced with the Windows syntax of a Unix path. The wildcard is used in the following way:

- {path.map.to.win.>}
- For the following input (?) --> /Volumes/Profiles_1/Users/Administrator/Pictures/helmut-IO.ico
- The output is going to be --> R:\Users\Administrator\Pictures\helmut-IO.ico

Note:

This functional wildcard returns the value from the path mapping table.

4.2.9. Path Map Auto ? Wildcard

This wildcard is replaced according to the operating systems representation of the input path ? (Requires matching path mappings defined in Helmut preferences be able to map across different platforms).

4.2.10. Path Map To JSON ? Wildcard

This wildcard is replaced with the JSON representation escaping any backslashes of input path. It can be used to point media files which requires escaped paths. The wildcard is used in the following way:

- {path.map.to.json.>}
- For the following input (?) --> R:\Users\Administrator
- The output is going to be --> R:\\Users\Administrator

4.2.11. Path Name ? Wildcard

This wildcard is replaced with the file name (incl. Extension) of a file path. The wildcard is used in the following way:

- {path.name.>}
- For the following input (?) --> C:\Users\Administrator\Pictures\helmut-IO.ico
- The output is going to be --> helmut-IO.ico

Notes:

Ensure that the render node or server has access to file. If the wildcard is on a server-side stream, the streams container needs to have access to the file path.

4.2.12. Path Parent ? Wildcard

This wildcard is replaced with the parent folder of a file path. The wildcard is used in the following way:

-
- {path.parent.>}
- For the following input (?) --> C:\Users\Administrator\Pictures\helmut-IO.ico
- The output is going to be --> C:\Users\Administrator\Pictures

Notes:

Ensure that the render node or server has access to file. If the wildcard is on a server-side stream, the streams container needs to have access to the file path.

4.2.13. Path Split ? Wildcard

This wildcard is replaced with the part of the file path that is defined. It is possible to define a part or a range to be used. This wildcard is used in the following ways.

- {path.split.#.?}
- For the following input (?) --> {path.split.0:2.\myserver\myfolder1\myfolder2\myfolder3}
- The output is going to be --> \myserver\myfolder1

Notes:

The split character is the \ (back slash) or / (forward slash)

The individual path selection can be done from left to right (with positive numbers i.e 0,1,2,3,4) or right to left (negative numbers i.e -1,-2,-3,-4) (example -1,-2)

When the : sign is used, it's possible to specify a custom path range for the left and right side (example 0:-1).

When a selection range is used (:), the number 0 can be used to set the left range on the first occurrence of the file path.

-1 sets the right side selection to the end of the file path.

Wildcard can be used on Windows and Unix file paths

File path needs to be formatted as \ (windows) or / (unix)

Advanced examples:

- Example 1:
 - Input --> {path.split.2.\myserver\myfolder1\myfolder2\myfolder3}
 - Output --> myfolder1
- Example 2:
 - Input --> {path.split.-3.\myserver\myfolder1\myfolder2\myfolder3}
 - Output --> myfolder1
- Example 3:
 - Input --> {path.split.2:-1.\myserver\myfolder1\myfolder2\myfolder3}
 - Output --> myfolder1\myfolder2\myfolder3
- Example 4:
 - Input --> {path.split.2:-2.\myserver\myfolder1\myfolder2\myfolder3}
 - Output --> myfolder1\myfolder2
- Example 5:
 - Input --> {path.split.0:-2.\myserver\myfolder1\myfolder2\myfolder3}
 - Output --> \myserver\myfolder1\myfolder2

4.2.14. Project Metadata? Wildcard

This wildcard is replaced by the value of a metadata entry that can be set via the metadata menu. The name of the metadata entry serves as input. The value can only be read out if there is an entry in the database for the object in the stream (project object). It is not enough for the metadata entry to exist in FX / IO. The wildcard is used in the following way:

- {project.metadata.?}
- For the following input (?) --> MyMetadataName
- The output is going to be --> MetadataValue

4.2.15. Stream Variable ? Wildcard

This wildcard is replaced with the value of a previously set, temporary variable. These temporary variables must be set within the stream and are only available while the stream is running. The wildcard is used in the following ways:

- {stream.variable.>}
- For the following input (?) --> MyStreamVariableName
- The output is going to be --> StreamVariableValue

4.2.16. String Split ? Wildcard

This wildcard is replaced with a part or a section of a string. The wildcard is used in the following ways.

- {string.split.>}
- For the following input (?) --> {string.split.0:2.I Want to Split this string}
- The output is going to be --> I Want to

Notes:

The split character is the blank space

Individual strings can be selected from left to right (with positive numbers i.e 0,1,2,3,4) or right to left (negative numbers i.e -1,-2,-3,-4) (example -1,-2)

When the : sign is used, it is possible to select a custom string range for the left and right side (example 1:-1).

When a selection range is used (:), the number 0 can be used to set the left range to the first string (example 0)

-1 sets the right side selection to the end of the string (example 3)

Advanced examples:

- Example 1:
 - Input --> {path.split.2.I Want **to** Split this string}
 - Output --> to
- Example 2:
 - Input --> {path.split.-3.I Want **to Split** this string}
 - Output --> Split
- Example 3:
 - Input --> {string.split2:-1.I Want **to Split this string**}
 - Output --> to Split this string
- Example 4:
 - Input --> {path.split.2:-2.I Want **to Split this** string}
 - Output --> To Split this
- Example 5:
 - Input --> {path.split.0:-2.I **Want to Split this** string}
 - Output --> I Want to Split this

4.2.17. String length ? Wildcard

This wildcard is replaced with the number of characters of defined string. The wildcard is used in the following ways.

- {string.length.>}
- For the following input (?) --> My string is this long
- The output is going to be --> 22

Notes:

Blank spaces are also counted

4.2.18. String Case To Camel ? Wildcard

These wildcards are removed from all spaces in a string and every first letter of a word is converted between upper and lower case. The wildcard is used in the following ways.

- {string.case.to.camel.>}
- For the following input (?) --> This wildcard formats the string like this
- The output is going to be --> thisWildcardFormatsTheStingLikeThis

4.2.19. String Case To Kebab ? Wildcard

This wildcard replaces all spaces in a string with hyphens. The wildcard is used in the following ways.

- {string.case.to.kebab.>}
- For the following input (?) --> This wildcard formats the string like this
- The output is going to be --> This-wildcard-formats-the-string-like-this

4.2.20. String Case To Lower ? Wildcard

This wildcard converts all uppercase letters of a string to lowercase. The wildcard is used in the following ways.

- {string.case.to.lower.>}
- For the following input (?) --> THIS WILDCARD FORMATS THE STRING LIKE THIS
- The output is going to be --> this wildcard formats the string like this

4.2.21. String Case To Pascal ? Wildcard

This wildcard removes all blank spaces of a string and converts the first letter of each word to uppercase. The wildcard is used in the following way.

- {string.case.to.pascal.>}
- For the following input (?) --> This wildcard formats the string like this
- The output is going to be --> ThisWildcardFormatsTheStringLikeThis

4.2.22. String Case To Snake ? Wildcard

This wildcard replaces all spaces in a string with underscores. The wildcard is used in the following ways.

- {string.case.to.snake.>}
- For the following input (?) --> This wildcard formats the string like this
- The output is going to be --> This_wildcard_formats_the_string_like_this

4.2.23. String Case To Upper ? Wildcard

This wildcard converts all lowercase letters of a string to uppercase. The wildcard is used in the following ways.

- {string.case.to.upper.>}
- For the following input (?) --> this wildcard formats the string like this
- The output is going to be --> THIS WILDCARD FORMATS THE STRING LIKE THIS

4.2.24. String Node Result ? Wildcard

This wildcard is replaced by the output value of a node whose NodeID is used. The node ID of each node that outputs an output value can be copied into the clipboard via the edit palette within the stream designer. See *WIP - H4 - Streamdesigner Guide - en.pdf*. The wildcard is used in the following ways.

- {node.result.>}
- For the following input (?) --> 152a8dc6-592b-43d8-874f-59f15f681a6a
- The output is going to be --> Node Output

4.2.25. String Node Result ? Wildcard

This wildcard is replaced by the output value of a node