Helmut 4 - Node

Guide

Software Documentation

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.

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). Fo example:

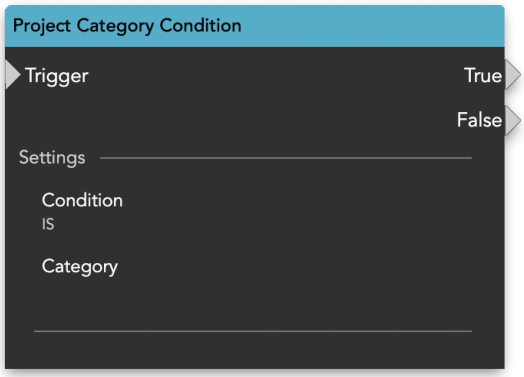

Figure 2.1. Condition Node

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:

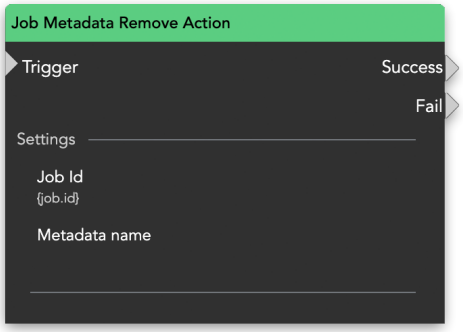

Figure 2.2. Action Node

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.

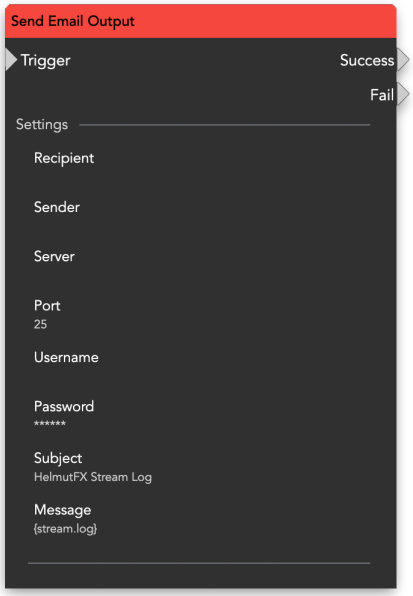

Figure 2.3. Output Node

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”.

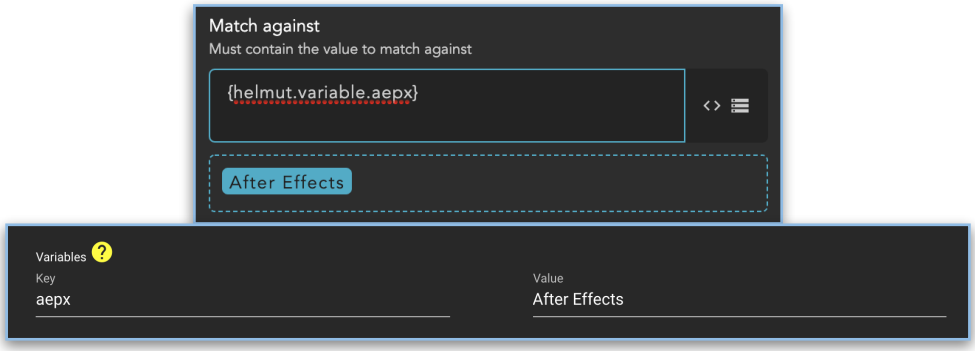

Figure 2.4. wildcard replacement

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.

3.1.2. 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.

3.1.3. 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.

3.1.4. File and Folder Conditions

3.1.4.1. 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.

3.1.4.2. 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.

3.1.4.3. 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.

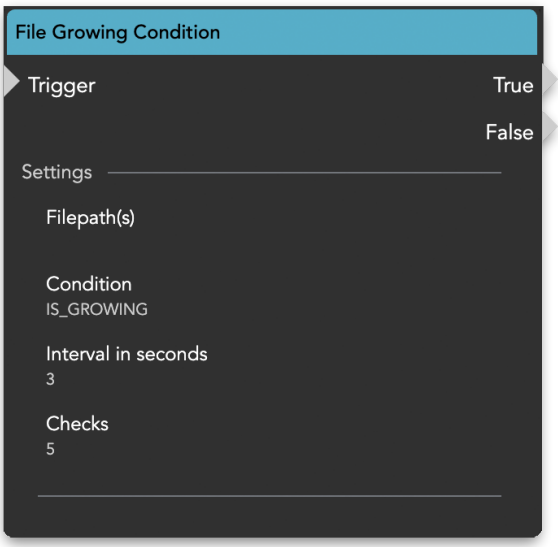

Figure 2.4. wildcard replacement

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.

3.1.4.4. 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.

3.1.4.5. 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.

3.1.4.6. 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.

3.1.4.7. 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.

3.1.4.8. 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.

3.1.5. OS Conditions

3.1.5.1. 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. Here it is possible to check whether the operating system is Windows, Unix, or MAC. This can be helpful, for example, for deciding which AME version is used for rendering, if the defined job profile can potentially be executed by different render nodes.

The Operating System Condition can be queried in every stream.

3.1.6. MediaInfo Conditions

3.1.6.1. 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.

3.1.6.2. 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.

3.1.6.3. 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.

3.1.6.4. 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 FPSt 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.

3.1.6.5. 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.

3.1.6.6. 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.

3.1.6.7. 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.

3.1.6.8. 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.

3.1.7. MISC Conditions

3.1.7.1. 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 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.

3.1.7.2. 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.

3.1.7.3. Regex Condition

With the help of the Regex 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.

3.1.7.4. 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.

3.2. 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.

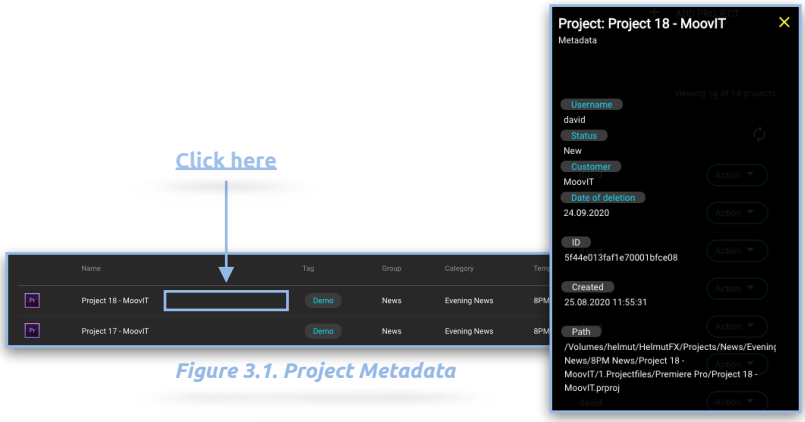

Figure 3.1. Project Metadata

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.

3.2.2. 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

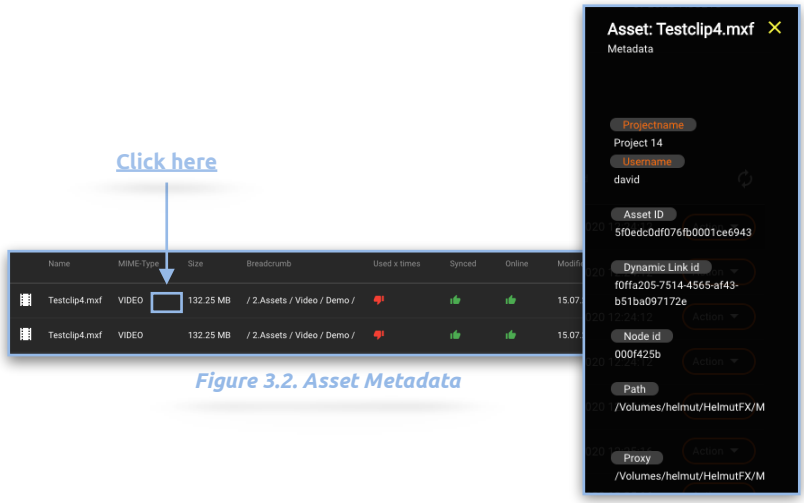

Figure 3.2. Asset Metadata

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 paypload 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 FFMEPG 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 behaviour 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.

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 Houskeeper 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 Houskeeper workflows, for example, a possible solution would be to create custom FX streams, which are used to create jobs in which Houskeeper 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 better still open. The node described here communicates with the invisible Helmut4 Premiere Panel Extension (installed automatically with the client), which executes the Extenscript 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)

3.2.3. 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. 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.

3.2.3.10. 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 a project that exists in Cosmo via a stream. 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, which, depending on which trigger was used for the stream, can also be filled via a wildcard {job.breadcrumb}. 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

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 a proxy to a project that exists in Cosmo and link the proxy to an existing high res file. If the high res file has not been added before trying to add a proxy, the node will fail. If a proxy has been added and attached to a high res file, it will be automatically synced into the Premiere Pro project (when opened) and thus the Premiere Pro Proxy Workflow (enabled via toggle switch within Premiere Pro) can be used.

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 Asset 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. 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 visualised in Cosmo but is available in the database.

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.

3.2.6. 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. 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.

3.2.6.4. 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.

3.2.6.5. 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.

3.2.6.6. 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.

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

3.2.6.7. 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.

3.2.6.8. 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.

3.2.6.9. 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.

3.2.6.10. Helmut Cloud Service Upload Action

With the help of the Helmut Cloud Service Upload Action, it is possible to upload one or more assets belonging to a project or a project itself to the Helmut Cloud Service via a stream. This service is offered by MoovIT GmbH and hosted by a cloud provider in Europe. (Contact us).

This makes it possible to work remotely and to automatically upload the files that are required via the cloud service in order to download them to the local computer (remote) in a further stream via the Helmut Cloud Service Download Action. This is an elegant solution to not route the traffic of the data through a VPN connection.

When used in a Housekeeper workflow in combination with the Cosmo Get Project Assets Action, it can be precisely determined which files of the project should be uploaded. Example:

It is conceivable to connect to the Helmut server in the company via a VPN tunnel and to connect a stream with the Open_Project trigger that executes the following actions:

- IP Condition to check whether the executing user is located in the company or is currently connected remotely.
- Cosmo get Project Asset Action to define the files to be uploaded
- Helmut Cloud Service Upload Action to upload every single file
- Helmut Cloud Service Download Action to download any file
- Cosmo Change Project Asset Action to change the paths in Cosmo to the new location on the local workstation
- Open File to open the project file, which now points to the paths on the local hard drive of the computer.
-

The rest of the work is done by the Helmut4 panel, which now links all assets in the project.

This workflow ensures that the user only has to open the project as he is used to. The process just takes a little longer.

3.2.6.11. Helmut Cloud Service Download Action

See 3.2.6.8

3.2.6.12. Helmut Cloud Service Delete File Action

With the help of the Helmut Cloud Service Delete File Action, it is possible to delete one or more files belonging to a project from the Helmut Cloud Service via a stream.

3.2.6.13. Helmut Cloud Service Delete Project Action

With the help of the Helmut Cloud Service Delete Project Action, it is possible to delete a project from the Helmut Cloud Service via a stream.

3.2.6.14. SwatIO Job Upload Action

With the help of the Job SwatIO Upload Action it is possible to publish an asset via a stream directly on or via SwatIO's Social Media Manager. There is the option of either transferring an asset to the platform (prepare), publishing it on a predefined date (publish), or publishing it directly on the desired channel (publish_now).

In addition, the title of the published article and a descriptive text can be defined. Both can be filled using metadata.

A possible use case is the publication of contributions on the common social media platforms directly via the Helmut4 panel in the form of an export stream. The handover of the bag and the text can either be left to the uers (metadata in the profile) or defined statically.

To operate the node, it is necessary to enter the corresponding channel ID of the social media platform. This can be found in the SwatIO Social Media Manager under Channels, in the respective URL of the channel.

3.2.6.15. 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}

3.2.6.16. 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.

3.2.6.17. 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.

3.2.6.18. 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 an a single asset that belongs to the project, the project will not be deleted, unless all assets belonging to the project have been deleted.

3.2.6.19. 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.

3.2.6.20. 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.

3.2.6.21. 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.

3.2.6.22. 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.

3.2.6.23. 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.

3.2.6.24. 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.

3.2.6.25. 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).

3.2.6.26. 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.

3.2.6.27. 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.

3.2.6.28. 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.

3.2.6.29. 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.

3.2.6.30. EVS Connector Action

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 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.

3.2.7.3. 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.

3.2.9. 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 standardised 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 behaviour 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.

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 MediaInfo as Json Action to retrieve the key value from one of these actions.

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. 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.

3.2.10.12. 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.

3.3.4. 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 (?) --> 1584748800000 [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 (?) --> 1584748800000 [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 --> 1584748800000 [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. 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

4.2.19. 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 Basename ? 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 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.

4.2.10. 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.

4.2.11. 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 formated 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

4.2.12. 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

4.2.13. 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

4.2.14. 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

4.2.15. 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

4.2.16. 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

4.2.17. 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

4.2.18. 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

4.2.19. 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

4.2.20. 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

4.2.21. 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

4.2.22. 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