SMEDGE

Administrator Manual

Smedge 2010

Table of Contents

Smedge Environment Variables

Variables that control Smedge functionality

These variables configure how Smedge itself operates. You must set these variables before starting a Smedge component process in order to have the value you have set recognized by the process.

SMEDGELIB	*API Only.* The base directory where the Smedge API is installed.
SMEDGE_BIN	*Linux only*. The actual directory where the Smedge binaries are installed. Normally, the Smedge start scripts take care of this for you, but if you want to be able to override the installation folder (for example, if you want to test a new version without affecting the working installation), you can override the path to the binaries with the value of this variable.
SMEDGE_DAEMON_PATH	*Mac daemon only.* Use this variable to override the location where the launchd plist file to control the daemon has been installed so that it can be properly shut down programmatically. If this variable is not set, the program will assume the plist file is in the defautl system daemon folder: `/Library/LaunchDaemons`.
SMEDGE_MACHINE_LOGS	Allows you to override the base folder where Smedge will write its own runtime log files. By default, this is in the local user directory for the user that started the component process. Using the environment variable will override the default, but using the command line flag -LogFolder will override the environment variable.
SMEDGE_MODULES	This is a semicolon separated list of folders to look for Smedge .SX Modules. The folders will each be recursively searched by the Module Manager during startup.
SMEDGE_OPTIONS_PATH	This is a semicolon separated list of folders to search when looking for INI files. The folders here will be searched before any default folders, but after any folders specified with the command line switches `-OptionsFolder` *folder*.
SMEDGE_UMASK	*Linux daemon only.* If you want to set a umask for the files created by Smedge when it is running as a daemon, you can set the umask value here. Smedge expects this value to be a 3 digit octal value corresponding to the bits you want umasked. For example: export SMEDGE_UMASK=022

`SMEDGE_USER`	*Linux daemon only*. This is the user account you want the Smedge 3 daemon process to run as. ***Warning***: if unset, the process will run as the user that started it. If you start it as root, then Smedge 3 Jobs will have potential root access to every machine on your network. **This could be very dangerous.**

Variables that Smedge can set for a work process

SmedgeEngine will set some more environment variables when it starts up. These variables will be valid from the context of any operation performed by the SmedgeEngine process, or by any child process it starts. This includes any form of work, no matter what Job classes it may use to implement its functionality.

`SMEDGE_ENGINE_ID`	The ID of the Engine on which the work was started
`SMEDGE_ENGINE_NAME`	The name of the Engine on which the work was started.
`SMEDGE_LOG_FOLDER`	The Smedge machine logs folder for SmedgeEngine.
`SMEDGE_PROGRAM_FILES`	The location where the Smedge program files, including the SmedgeEngine exectuable, reside.

Smedge work that spawns a child process (anything that is derived from ProcessJob, technically) will create several additional environment variables that the work process can use to access information from the Smedge system without having to pass the data directly via the command line. See the ProcessJob parameters list for more information. These variables are only set for a child process started from ProcessJob::Execute. They are not available from the context of the SmedgeEngine process itself, unlike the variables above.

`SMEDGE_COMMAND_LINE`	The command line that was used to start the child process.
`SMEDGE_JOB_ID`	The ID of the parent Job
`SMEDGE_JOB_NAME`	The name of the parent Job.
`SMEDGE_WORK_ID`	The ID of the work unit.
`SMEDGE_WORK_NAME`	The name of the work unit
`SMEDGE_WORK_PARAMETERS`	Any custom parameters that you have exported from the ProcessJob $(EnvironmentParameters) parameter of the Job.

Restrictions

Smedge includes a restriction system that allows administrators to limit the SmedgeGui functionality for end users. For example, you may want to limit users to only being allowed to modify their own Jobs. The restriction system allows this.

You can configure Restrictions in the Configure Master dialog. For more information on how this window works and how to access it, see the SmedgeGui reference in the User Manual. Note that restrictions currently only affect SmedgeGui operation. Entering "Administrator Mode" will override any Restrictions set, and allow complete SmedgeGui functionality. The following is a list of the currently available restrictions and what they do:

Change Job	Users cannot change any parameter of a job once it has been created.
Configure Connection	Users cannot change their local user preference for how SmedgeGui to connect to a Master .
Configure Connection.ini	Users cannot use SmedgeGui to configure the Connection.ini file for how any Smedge client application on the system can connect to the Master.
Delete Job	Users cannot delete Jobs from the system.
Edit Engine	Users cannot modify any Engine settings, Engine Product Options, or what Pools an Engine belongs to. Essentially, the Edit Engine window can no longer be accessed.
Job Priority	Users cannot set or change the priority of any Job.
My Jobs Only	Users can only control or modify Jobs that they have created. This restriction will also remove the ability to customize the user name that is set as the Job "Creator", so that users cannot defeat the system just by changing the user name in the SmedgeGui options.
Process Control	Users cannot start, stop, install or remove the Master and Engine components from SmedgeGui.
Reset Failures	Users cannot access the Reset failures commands to reset Job or Engine failure counts.
Stop Work	Users cannot stop work from any Job.
Submit Job	Users cannot submit new Jobs to the system using SmedgeGui.
VNC	Users cannot access the remote desktop viewing commands.

RLib INI File Syntax

RLib INI files build on the basic section, key and value scheme. Section names and Key names are stored in a case insensitive manner, so watch out because “SECTION,” “Section,” and “section” will all refer to the same section.

Sections are delimited by text placed in square brackets:

```
[ Section Name ]
```

The square brackets will be removed, and any whitespace between the brackets and the text will be trimmed. Spaces are allowed inside the section name. A section name must be contained on a single line. If you have multiple sections with the same names, the values will be added to the same section, or will overwrite existing values with the same key.

Data is stored in sections in a key equals value format:

```
Key Name = Value
```

The equal sign character is the important token. Without this character, the data will fail to load correctly. Whitespace around the equal sign will be trimmed off, along with any whitespace at the end of the data. You can surround your value in quote marks, which will allow you to have actual whitespace as a value. If you do this, the quote marks will be removed. For example:

```
Space = " "
```

A line like this will set the value of a key called “Space” to be a single space character.

Unlike sections, keys can span multiple lines. Text that is found without an equal sign will be appended directly on the end of the last found value. No extra space is added.

You can insert comment lines by making the first non-whitespace character of the line be either the semicolon (`;`), pound sign (`#`), or single quote (`'`):

```
# This is a comment line!
; So is this!
' And this too!
```

RLib loads the whole file at once, and provides access to the data loaded through the IniFile class. This class will be documented more fully in the RLib API documentation. What this means, however, is that you cannot have multiple keys in the same section with the same name. The last one read will override any previous values.

Alternate file locations

In the Smedge cross-platform system library, files can use a system of alternate locations to allow you to create and change your default configurations more easily. The alternate file location system is built into the basic File management system of the cross-platform library that underlies all Smedge operation, but how it is used depends on your use of the Smedge API.

When you are trying to just open an file, if the file is not found in the directory specified, or if no directory is specified as part of the name, Smedge tries to find the file in other standard locations. The locations are searched in this order, and as soon as a file with the same name is found in any of those locations, that file is loaded, and the searching stops.

The folders searched are:

1. The folder specified in the file path, if any

2. The current user's options folder for the component application that is trying to use the file.
   On Windows: `C:\Documents and Settings\`*User*`\Application Data\Uberware\`*Component*
   On Linux: `~/.Smedge/`*Component*
   On Mac: `~/Library/Smedge 3/`*Component*

3. The current machine's options folder for the component application that is trying to use the file.
   On Windows: `C:\Documents and Settings\All Users\Application Data\Uberware\`*Component*
   On Linux: `/etc/smedge3/`*Component*
   On Mac: `/Users/Shared/Smedge 3/`*Component*

4. The Smedge 3 program folder.

5. Any folders specified with the `-OptionsFolder` *folder* command line flag. If more than one folder is specified, they are searched in the **reverse** of the order they appear after the command line flag.

6. Any folders specified in the `SMEDGE_OPTIONS_PATH` environment variable. If more than one folder is specified, they are searched in the **reverse** of the order they appear in the environment variable.

The first file found is the one read, and the search is stopped as soon as a readable file is found. Using the API, if the file is found in one of the alternate locations (any location from 2 through 6) it is possible to configure the file to copy the file from the alternate location to the originally specified path. See the API documentation and headers for more information.

Overloadable Options Files

The program options INI file has more advanced “overloading” functionality. Instead of loading the file only once, it actually loads the file multiple times from several locations, allowing you to specify common default option values that can be specifically overridden for a specific machine or user. This system is available to any file operation that uses the OptionsFile API system. Normally, this is limited to files that contain the options and settings for a component application.

This system reads the file in the following order:

1. Every folder specified in the `SMEDGE_OPTIONS_PATH` environment variable. If more than one folder is specified, they are searched in the order they appear in the environment variable.

2. Every folder specified with the `-OptionsFolder` *folder* command line flag. If more than one folder is specified, they are searched in the order they appear after the command line flag.

3. The Smedge 3 program folder

4. The current machine’s options folder for the component application that is trying to use the file.
   On Windows: **`C:\Documents and Settings\All Users\Application Data\Uberware\`***Component*
   On Linux**:** **`/etc/smedge3/`***Component*
   On Mac: **`/Users/Shared/Smedge 3/`***Component*

5. *For Shell components only:*
   The current user’s options folder for the component application that is trying to use the file.
   On Windows: **`C:\Documents and Settings\`***User***`\Application Data\Uberware\`***Component*
   On Linux: **`~/.Smedge/`***Component*
   On Mac: **`~/Library/Smedge 3/`***Component*

If a file with the same name exists in more than one of the locations, it will be read from each location where it is found. If an key is set in an earlier file, but not in a later file, then that key’s value will be used as the “default” value. If the key exists in more than one file, then the file most recently read file will specify the key’s value. Any values unspecified in any file read will use the built-in default values for whatever element of the Smedge system is trying to access the options (for example values specified for a Virtual Module will come from the Virtual Module definition file, and values specified for a component application’s behavior will come from the hard coded defaults compiled into the application).

Note that the Master and Engine do not use the user’s options folder for storing options. Master and Engine options are always stored for the machine as a whole, so location #4 is where the customized options are always stored.

Variable Substitution

Smedge includes a powerful variable substitution system that is used in just about every component application and Module in some way. Most often, this system allows you to generate a command line to spawn a work process using data from the Job to replace variables in a standard command line syntax. However, the system is also used by the Herald to extract Job data when performing an action, and can be used for event commands attached to Jobs, and is used in many other places throughout the Smedge system.

The information in this section is primarily directed towards users that wish to create custom Virtual Modules. However, the parameter system is also used when you are creating Job Event commands, and is also commonly used in the Herald to allow access to information from the Job or Work that triggers the notification. For example, if you want to send an email when a Job finishes, you probably want to include the Job name in the email, and the Parameter system allows you to do that easily.

Syntax

The syntax for accessing a parameter is to start with a dollar sign, then surround the name in parentheses. You can optionally attach one or more commands to perform on the parameter before it is substituted back into the resulting string. The full syntax is:

```
$(ParameterName.Commands…)
```

The entire text from the dollar sign to the closing parenthesis will be replaced with the actual value from the Job object for the parameter named. The value may come from one of many places depending on the settings of the Job parameter being accessed:

1. If the ParameterName names a valid parameter from the Product, one of the following three sources will be used:
    a. If the parameter names a Job parameter, and that parameter has a non-empty value for the specific Job being accessed, that value will be used.
    b. If the parameter names a Product option for the application type accessing it (e.g. an Engine Option for access by the Engine), and that option has been configured in the options for that component application, that value will be used.
    c. Otherwise, the hard coded default value for the parameter will be used.
2. If the ParameterName names an environment variable, that variable’s value will be used.
3. If the name cannot be found, the variable will just be removed.

You can access any parameter value using its internal name. Be aware that this process is recursive: if you access a parameter that has data that uses this syntax, that value will also be parsed and replaced with data before being replaced into the command line. Smedge will check for direct recursion, where the parameter name appears inside the parameter value itself. However, it is possible to set up a chain of two different parameters that become mutually recursive. This will crash Smedge. Avoid doing this.

You can set part of the variable substituted string to be dependent on a non-empty value found for a parameter by enclosing it in square brackets. The entire text inside the square brackets will be dropped if the parameter inside the brackets cannot be found or is an empty string. For example, say you have this:

```
[-left $(Left) ]
```

If a parameter named **Left** is found and has the value **100**, the block will be replaced with this text:

```
-left 100
```

If, however, the parameter **Left** cannot be found, or has is an empty string, the entire block will be left out of the formatted string.

Parameter Commands

The optional commands will be processed after the value is determined and before it is substituted back into the string. Commands are denoted with a period and then the command name, one of the names in the table below. You can chain commands by adding another period and another command name. Commands are processed from left to right. All commands will be processed before the final result value is substituted back into the result value.

These are all the commands currently available. The command processing will happen regardless of the type of the parameter, so the result may be unexpected if you use it on the wrong type of parameter.

Command	*What it does*
Absolute	If the value refers to a filename or a relative path, it will make that path into the absolute path where the file will be created on your system.
CopyLocally	Copies the file to a local temp folder and returns the path to the local copy. If the source file exists and either has not been copied, or is newer than the copy, it will be copied immediately upon the execution of the command. If the command is accessed as part of a parameter for a Job, all files copied will be deleted when the Job finishes. Otherwise the copied files will remain in the temp folder. The folder copied to is $TEMP/smedge3/LocalCopies/*job-id*.
CutRoot	Attempts to return everything except the root drive part of a full file path or directory
CutExtension	Attempts to return everything except the extension of a file path

Command	*What it does*
Dequote	Makes sure that there are no double quotes around the value, even if there is a space in that value.
End	Returns anything after the last **–** or **,** character, or the whole value if there is no – or ,
Enquote	If there is a space character in the value, it will be surrounded by double quote marks, allowing the OS to treat it as a name with a space in it.
Extension	Attempts to return the extension part of a file path
File	Attempts to return the filename part of a full file path
Hex	Converts the value to a hexadecimal number
Local	Checks the path to see if it can be converted to the local platform using the Path translation system.
MakeLower	Converts the value to all lowercase letters
MakeUpper	Converts the value to all uppercase letters
Pad	Converts the value to a four digit padded, signed number
PadUnsigned	Converts the value to a four digit padded, unsigned number.
Path	Attempts to return the directory part of a full file path
Root	Attempts to return the root drive part of a full file path or directory
Safe	Converts any characters that are unsafe for a filename into _ characters.
Start	Returns anything before the first **–** or **,** character, or the whole value if there is no – or ,
WordUpper	Capitalizes only the first letter of each work (separated by spaces)

For example, to access the directory part of a scene parameter you could use this syntax:

```
$(Scene.Path)
```

To access the filename without the directory part, you could use this syntax:

```
$(Scene.File)
```

To access the start and end frames of a sub-range, you could use these:

```
$(SubRange.Start) $(SubRange.End)
```

This converts a plain filename into the absolute path to that file, and then puts quotes around it if that absolute path happens to have a space character anywhere in it:

```
$(Scene.Absolute.Enquote)
```

Common Parameters

The following tables present all of the parameters available in Smedge currently. Note that Smedge uses a "class hierachry" system to add Job functionality. See the reference for each specific Product to see which of these common classes are included for that Product. Authors of Compiled Modules, using the Smedge API, may not make use of the full class hierarchy, if they don't need some functionality, so some of these parameters may not be available for these custom Modules. When in doubt, check with the author of the custom Module. Also, check the Products chapter for more information about each specific Product that currently ships with Smedge.

Name is the parameter name you can use to access this parameter. This is the text that goes inside of the $(*Name*) syntax, or that you use as a parameter with the **Submit** commandline shell to submit jobs, in a –*Name* syntax (See the documentation for Submit in the User Manual for more information). **Type** is the type of value that Smedge expects for this parameter. See Parameter Types for more information about what the type means. **Get** means that you can use this parameter to get a value with the $(*Name*) syntax. **Set** means that you can set this value with the **Submit** command line shell (or programmatically with the Smedge API). **Meaning** gives a brief description of how the parameter will be used. **Default** is the value that will be filled in for the Job if you don't supply it at submission time. Parameters in red *must* be supplied when you submit the Job.

Job

The Job class provides the basic, common functionality for all jobs and workers. Every Job and Work unit has at least the Job information.

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
CPUs	Int	X	X	The number of CPUs/threads assigned to this Job	0
Created	Time	X	X	The time when the Job was created	*Set by Master*
Creator	Text	X	X	The Job's creator string value	*Set by Shell*
CurrentDate	Text	X		Gets the current date formatted as a string. You can optionally provide a formatting string for the date by putting a colon and then the formatting string before any parameter commands or the closing parenthesis. See the RLib Time API documentation for the time and date formatting syntax. The default is %d-%b-%y	
CurrentTime	Text	X		Gets the current time formatted as a string. You can optionally provide a formatting string for the date by putting a colon and then the formatting string before any parameter commands or the closing parenthesis. See the RLib Time API documentation for the time and date formatting syntax. The default is %H:%M:%S	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
DeleteJobEvt	Text	X	X	The event command to execute when the Job is deleted. This command will have parameters expanded and then will be executed asynchronously by the Master.	
FirstWorkEvt	Text	X	X	The event command to execute when the first work from this Job is started on an Engine. This command will have parameters expanded and then will be executed synchronously by the Engine.	
ID	ID	X		The Job's unique identification number	
JobAssignWorkEvt	Text	X	X	The event command to execute when the Master is about to assign work to an Engine. This command will have parameters expanded and then will be executed synchronously by the Master.	
JobFinishedEvt	Text	X	X	The event command to execute when all work from a Job has finished or been permanently canceled. This command will have parameters expanded and then will be executed synchronously by the Master.	
JobFirstStartedEvt	Text	X	X	The event command to execute when the first work from this Job is started. This command will have parameters expanded and then will be executed synchronously by the Master.	
JobLocalFolder	Dir	X		The Job specific local temp folder where files are copied by the **CopyLocally** parameter command	
LogDir	Dir	X		The application's log directory	
MachineDir	Dir	X		The machine-wide options folder	
MachineName	Text	X		The name of the local computer	
MachineNumber	Int	X		A number hash based on the local computer's name (there is a very small, but non-zero chance that two different machines' names will hash to the same value).	
Name	Text	X	X	The Job's name	
Note	Text	X	X	The Job's note	
OvertimeKill	Float	X	X	If a worker goes this many times over the average time for workers from this Job, the worker will be timed out and requeued. (Set to 0 to disable).	15
Parent	ID	X		The Parent Job's identification number	
ParentName	Text	X		The Parent Job's name, if it can be found	
Pool	ID	X	X	The Job's pool identification number	Whole System
Priority	Int	X	X	The Job's priority value	50
SmedgeDir	Dir	X		The Smedge program folder	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Status	Int	X		The Job's current status code	
StatusAsString	Text	X		The Job's current status as in a human readable format	
SystemID	ID	X		The unique system ID.	
TempDir	Dir	X		The system's TEMP directory	
Type	ID[1]	X	X[2]	The Job's type identification number	
TypeString	Text	X		The Job's type's Name string	
UsageLimit	Int	X	X	The limit for this job to have outstanding	-1
UserDir	Dir	X		The user's options folder	
WorkAssignedEvt	Text	X	X	The event command to execute when the Master has asked the Engine to start work from a Job. This command will have parameters expanded and then will be executed synchronously by the Engine.	
WorkEngine	ID	X	X	The ID of the Engine that is assigned to perform the Work. NULL if the work has never been assigned to an Engine or for Parent Jobs.	
WorkEngineName	Text	X		The name of the Engine that is assigned to perform the Work. Uses the application's GetEngineByID() method to get Engine data associated with the ID from **WorkEngine**. Returns "No Engine" for NULL IDs, and the ID as the name if the name cannot be found from the application.	
WorkFinishedEvt	Text	X	X	The event command to execute when the Engine has finished a work unit. This command will have parameters expanded and then will be executed asynchronously by the Engine	
WorkFinishedSuccessfulEvt	Text	X	X	Called immediately after WorkFinishedEvt for successful work. This command will have parameters expanded and then will be executed asynchronously by the Engine	
WorkFinishedUnsuccessfulEvt	Text	X	X	Called immediately after WorkFinishedEvt for unsuccessful work. This command will have parameters expanded and then will be executed asynchronously by the Engine.	
WorkParameterChangedEvt	Text	X	X	The event command to execute when the Engine has detected a change in a parameter for a currently executing work unit. This command will have parameters expanded and then will be executed asynchronously by the Engine.	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
WorkPostExecuteEvt	Text	X	X	The event command to execute after the work executes but before the result is sent to the Master. This command will have parameters expanded and then will be executed synchronously by the Engine.	
WorkPostExecuteSuccessfulEvt	Text	X	X	Called immediately after WorkPostExecuteEvt for successful work. This command will have parameters expanded and then will be executed synchronously by the Engine.	
WorkPostExecuteUnsuccessfulEvt	Text	X	X	Called immediately after WorkPostExecuteEvt for unsuccessful work. This command will have parameters expanded and then will be executed synchronously by the Engine.	
WorkStartedEvt	Text	X	X	The event command to execute when the Engine has accepted a work assignment and is about to start working. This command will have parameters expanded and then will be executed synchronously by the Engine.	

[1] When setting the Type, you can generally use any number of possible values, including the ID, the Product's Name, or any of the defined Shortcuts for that Product. See the Products chapter for a reference of all values you can use for every Product currently distributed with Smedge.

[2] Once a Job has been created, you cannot modify the Type.

ProcessJob

ProcessJob provides the data and services necessary to have a Product that performs work by launching a separate process to do the work. This is how most Products work in Smedge. The alternative is to have the SmedgeEngine process actually do the work itself as code directly compiled into the Module. For example, the Large File Transfer job actually does not use a separate process to do the work, but simply does the requested operation directly in SmedgeEngine.

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
CheckReturnValue	Bool-Override	X	X	Allows you to bypass the check on the code returned from the spawned child process. When enabled, any non-zero return value is interpreted as an error, resulting in the work being requeued.	Engine Default (yes)
DetectErrors	Bool-Override	X	X	Determines if the process output is monitored for error messages	Engine Default (yes)

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
EnvironmentParameters	Text	X	X	This value will be expanded with any data from the Job and exported to the environment as the **`SMEDGE_WORK_PARAMETERS`** environment variable. **Note:** If this value expands to an empty string, the environment variable will not be set (or unset)	
ErrorIgnores	TextList	X	X	A list of strings to look for in a line of output that has been detected as an error by the containing one of the ErrorStarts values. This allows you to specify errors that can be safely ignored by the system.	
ErrorStarts	TextList	X	X	A list of strings to look for in a line of output that can be used to detect that an error has occurred in the processing of a work unit. If any of the elements of this list is found in a line of output and none of the ErrorIgnores values are also found in that line, the work is assumed to have failed, and it will be immediately terminated and requeued for later processing.	
Executable	Path	X	X	The rendering executable program file. You normally won't need to access this yourself. It's handled automatically by ProcessSequence.	
IdleTimeout	Float	X	X	The number of seconds that a spawned child process is allowed be running without consuming any CPU resources before it is considered timed-out, and requeued.	300
OutputPath	Dir	X	X	The directory in which saved captured output files will be saved.	
OutputPeer	Text	X		The IP address and port where the work is serving the output from.	
ProcessPriority	Choice	X	X	One of "Normal Priority" "Low Priority" or "Idle Priority"	Normal Priority
Password	Password	X	X	The password that will be used to gain access to requested resources, if needed.	
Resources	TextList	X	X	*Windows only.* A semicolon separated list of Drive=Share pairs that the work unit will try to ensure are available before starting work.	
ShowProcess	Bool-Override	X	X	Determines whether the SmedgeEngine should try to spawn the process visibly, instead of hidden from view. This may not have an effect if the spawned process does not have any kind of user interface, or if SmedgeEngine is running as a background process on the machine.	Engine Default (no)

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
StartDirectory	Dir	X	X	The directory that will be made the currently active directory when the process is started.	
Username	Text	X	X	The username that will be used to gain access to requested resources, if needed.	

SequenceDistributor

SequenceDistributor is a Job Distributor. This is the component of a Job that the Master uses to actually divide the job up into work units to be distributed to the Engines. SequenceDistributor distrbutes work as subsets from a range. In Smedge, this generally means a range of frames to render, and the subset is the number of frames that will be rendered at one time by one machine in one work unit. Note, however, that SequenceDistributor simply uses the set range you supply, and does not know or care about how that set range actually applies to the product in question.

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
JobSummary	Text	X		Shows the SubRange of the Job or the Range if the SubRange is empty.	
PacketSize	Int	X	X	The packet size used by the parent Job to break the work up. Note that there may be fewer frames in the range than the packet size allows, if the division came to the end of a sequence.	1
Range	Text	X	X	The entire range of the parent job. This can be a complex range string including both – and , to separate elements.	
RenumberBy	*Text* *The type currently needs to be set to Text to work correctly, but should generally be used as an integer value	[3]	[3]	*If it exists in a derived class, this value will be used to correctly calculate the renumbering. This parameter does not exist in this class, but will be used if it exists in your Virtual Module.*	[3]

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
RenumberStart	*Text* *The type currently needs to be set to Text to work correctly, but should generally be used as an integer value	[3]	[3]	*If it exists in a derived class, this value will be automatically updated for each worker to correctly allow you to renumber the sequence. This parameter does not exist in this class, but will be used if it exists in your Virtual Module.*	[3]
SequenceBy	*Text* *The type currently needs to be set to Text to work correctly, but should generally be used as an integer or floating point value	[3]	[3]	*If it exists in a derived class, this value will be used to correctly calculate the renumbering. This parameter represents a different sequencing than just 1 at a time. For example, you may want to render every other frame from a sequence. To do so, add a parameter to your Job class with this name. This parameter does not exist in this class, but will be used if it exists in your Virtual Module.*	[3]
SubRange	Text	X		The entire range of the work, formatted as a string in **start-end** format. There is no space in the formatted range. If the work has only one frame, it will be returned the same as if you used **SubRange.Start** or **SubRange.End**.	
WorkFinished	Text		X	The text should indicate the range of the work that has just finished.	

SliceDistributor

SliceDistributor is a Job Distributor. This is the component of a Job that the Master uses to actually divide the job up into work units to be distributed to the Engines. Slice distributor is designed to divide a single frame render into slices that are then combined back together into the final result. It distributes a worker for each slice, then a final worker that does the combination processing.

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
JobSummary	Text	X		The repetition of the child Work or the number of slices of the parent Job	
Slice	Int	X	X	The repetition number of a specific child Work	
Slices	Int	X	X	The number of slices to divide the parent Job into	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
WorkFinished	Text		X	The value should indicate the repetition of the work that has just finished.	

RenderJob

RenderJob is designed around the concept of a Job that renders an image or image sequence from an image processing or generation tool. It is optimized for 3D design and animation and 2D compositing tools, but it can be used to manage most related systems as well, such as file conversions, cache export, simulations, etc. The terms may differ, but the concepts of these processes are similar, and can fit into this paradigm.

The key elements it provides are the concept of the "scene" that is being rendered, and the "images" it produces as a result. It also provides some common error detection associated with the operation of the process and the results it produces. The process will be spawned with parameters to process the "scene" using the distributor's sequencing information, and will make the "image" results available in the system, if it can.

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
CheckImages	Bool-Override	X	X	An option to have the Job try to validate any detected frame filenames before reporting a successful completion.	Engine Default (yes)
DeleteBadImages	Bool-Override	X	X	An option to have any image files that are detected but that fail the image check test (if enabled, see **CheckImages**) deleted at the end of the work unit.	Engine Default (no)
DetectImageFormat	Bool-Override	X	X	An option to have the Job try to automatically detect the image formats from detected rendered image filenames.	Engine Default (yes)
ImageDir	*Dir*	[3]	[3]	*If it exists in a derived class, this value will be prepended to any detected image filename if they are not already absolute. If this value is empty or cannot be found, RenderJob will try to prepend the start directory to any relative filenames. This parameter does not exist in this class, but will be used if it exists in your Virtual Module.*	[3]
ImageFile	File	X	X	Get will return the last detected image file, Set will append the given file to the list of detected files	
ImageFileList	Text	X		Returns the entire list of detected image files, enquoted and separated by spaces. Useful, for example, for passing the entire list of image files to another application, for display or compositing.	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
ImageFormat	TextList	X	X	A list of the filename path and formats. The format string is kept in printf format, where the frame range is replaced with a formatting string like %d. This parameter is used by Smedge Shells, but its value does not affect the actual rendering output. You must supply parameters specific to the individual Products to actually configure how the product will format the rendered frame files.	
Scene	File	X	X	The scene file that the user selected to process.	
SequenceName	Text	X		Always returns "$(SequenceName)" as an unformatted string. This is used internally by Shell programs as a placeholder for the view sequence command.	

[3] Access and default depend on the implementation in the derived Product.

Dynamic Products

Several Modules use this simple Dynamic Product creation system to make it easy to support multiple versions of the software on your Engine at the same time. The Products are defined in an INI file with the same name as the Module file. For more information about the RLib INI file format, see the chapter on RLib INI File Syntax.

The Module will always create at least one product, with the ID given for that Product in the Products chapter. You can override the values that define that Product, and you can also add new Products. To override the existing Product values, just specify that Product's ID as the INI file section. For new Products, make sure to use a new unique ID. You can use the **`uidgen`** program included in the Smedge distribution to generate UIDs.

Products are defined by creating a new section in the file with the Product's ID. You can then optionally override any or all of the parameters that define the Product. Any parameters you do not override will use the default value that the Module provides for the default Product.

`[` *Product ID* `]`

`Name` = *Product Name*
`Alias` = *comma separated list of short cuts used for* `Submit -type`
`DefaultEnabled` = `yes` *or* `no`
`Executable` = *full path to the executable for this particular Product*
`ShortHelp` = *short help message*
`LongHelp` = *longer help message*
`OverrideDefaults` = *A semicolon separated list of {Name} = {Value} pairs that will change the default value of the named parameter for this Job type.*
`OverrideFlags` = *A semicolon separated list of {Name} = {Comma separated list of flags} pairs that will change the flags of the named parameter for this Job type.*

Currently this system is used by these Modues: AfterEffects.sx, Lightwave.sx, Max.sx, Maxwell.sx, MentalRay.sx, Nuke.sx, which supply the After Effects, Lightwave, 3D Studio Max (and related), Maxwell, MentalRay Standalone, and Nuke Products.

Maya Products

The Maya.sx Module supports a hybrid customization of its own, that provides a subset of the entire Virtual Module functionality provided by the Process Sequence.sx Module, that is specifically oriented towards using the Maya integrated renderers. As with Modules that support Dynamic Products, the Maya module will load the Product definitions from an INI file. The Maya Module also adds the ability to customize the options shown in the “Render Overrides” parameter. **Note:** before Smedge 3 version 2.3, this parameter was called “Extra Parameters” and it is still accessed through the internal name $(Extra).

The Smedge Module definition file also allows you to override how the Maya module will look for detectable image files from the Maya output stream, and you can override the default values and flags for any other parameter provided by the Maya module.

Smedge ships with a Maya.ini file in the Modules folder that includes definitions of all of the renderers that are supported through Maya 2008, and several third party renderers that have been integrated in, like RenderMan for Maya and finalRender for Maya. You must supply the values shown in bold.

`[` *Product ID* `]`
`Name` = ***The name of the product***
`Shortcuts` = ***Comma separated list of alternate shortcut names***
`Help` = ***Short help text***
`LongHelp` = ***Long help text***
`CommandLine` = ***The command line parameters to send to the Render.exe executable***
`DefaultEnabled` = **`yes`** ***or*** **`no`**

`Extra` = *The list of extra parameter internal names*

`ImageCheck` = *Semicolon separated list of image filename search tags in this format:*
`[`*Line Start Text*`] [/ [`*Image Filename Text End*`]]`
Either the Line Start Text *or* Image Filename Text End *parameters can be empty for any entry.*
The detected image file will start with the specified text, and end before the specified end text
It will also automatically have whitespace trimmed and be dequoted.

`ImageNotEnd` = *Semicolon separated list of text strings that will be used to exclude lines from the Image detection system. Excludes any lines that include any of the given text strings anywhere in the line.*

`ErrorStarts` = *Semicolon separated list of text strings. If a line of output starts with one of these Smedge assumes that means an error has occurred. And aborts the work unit.*
If you do not include this value, it will default to the single entry `Error:`

`ErrorIgnores` = *A semicolon separated list of text strings. If one of these strings is found in a line of output detected as an error because of an Error Start String, that line will be ignored, and the work will not be aborted or marked as failed.*

The Parameters listed in the **Extra** field are also defined in the file. They are defined in the same format as Parameters are defined in Virtual Module files. See the Custom Parameters in the Virtual Module chapter for more information. The default Maya.ini file includes definitions for every parameter available for every renderer included with the current release of Maya, available for reuse as needed for each Product. For more information on what these parameters do, see the Maya software documentation.

Virtual Modules

Smedge 3 includes a module that allows the creation of custom render types via a relatively simple text file. The Module is called **ProcessSequence.sx** and the text files used are called **Virtual Modules**. Virtual Modules are defined in a text file with a **.PSX** extension.

Virtual Modules are derived from the RenderJob class in SmedgeLib. RenderJob is designed to control a third party rendering application via a command line interface to render a sequence of frames. This lineage gives your virtual modules nearly all the functionality you need not only to start and control the rendering of a sequence of images, but to detect and examine the rendered image files, to make resources available at work execution time, and to capture and save the output from the spawned program. Virtual Modules can handle both single scene rendering and one scene file per frame rendering.

When a ProcessSequence Module is loaded, it scans the directory it is in for all files with the .PSX extension. Each .PSX file is designed to hold a single virtual Job type. Every file found by the module will be controlled by that module. If a type is loaded that has a conflicting ID with an already loaded or otherwise existing type, it will simply be ignored. You should only have one copy of the ProcessSequence module in a given directory. Otherwise, if multiple copies of the module are loaded, they will find all the same virtual modules without actually loading any of them.

Because ProcessSequence does not recurse directories, it could be useful for testing purposes to have a copy of the ProcessSequence module in a child directory with just the virtual module you are working on. This way, it will be run by a separate Module than any others. It would be a good idea to rename this module if you do this, which will make it much easier to dynamically load and unload.

PSX Files are simple text documents in an INI style. ProcessSequence uses RLib's IniFile class to handle the actual loading of these files. See the RLib INI File syntax section for more information. Note that ProcessSequence does not use the standard RLib alternate file location system to locate Virtual Module Files. It will only scan for files in the same directory as the compiled module file itself.

Parameter Types

Every parameter has a "type" that gives an indication of how this parameter is expected to be used. There are many types available, but not all types will make sense for all applications. The parameters are passed around as text strings, but they may not be stored that way. There may be some validation associated with some types when you set their value.

Type	*Elements*	*Description*
`Alternate`	Alternate	A combination of a Boolean value that will result in one of two names for the parameter. Useful for alternate switches for a toggle value where two different names are used for the choices. For example: [-blur/-noblur]

Type	*Elements*	*Description*
`Bool`	Bool	A true/false value, with customizable formatted value strings
`BoolOverride`	BoolOverride	Adds a third possible state to a Boolean value for the Job type: Engine Default. If the Job parameter is set to this value, then the Engine's value will be used. This type requires that the parameter is both a Job parameter and an Engine option to make any sense.
`Choice`	Choice	Select one of a list of strings. You can have a different value for display and for use as a formatted parameter.
`Dir`	Directory	A text field interpreted as a directory.
`DirList`	Dir	A list of directories separated by semicolons
`File`	File	A text field interpreted as a file. Has a prompt text and a filter. The prompt is displayed to the user by Shells as part of the request to find a file on the local system. The filter is a file selection filter like: `C++ Files (*.cpp, *.cxx)|*.cpp;*.cxx|{All Files}`. If no filter is supplied, the filter will default to: `{All Files}`. If the string includes the single filter `{All Files}` (case insensitive and including the {}s) an default for the local system should be added by Shells appropriate to the local platform. If the string includes the single filter `{Executable Files}` (case insensitive and including the {}s) Shells should add the appropriate filter for the local platform for locating executable files.
`FileList`	File	A list of files separated by semicolons
`Float`	Preset	Like **Text**, but interpreted to be a floating point number
`ID`	Preset	Like **Text**, but interpreted to be a UID in 8-4-4-4-12 hexadecimal format
`Info`	Common	Not a parameter, but an extra bit of textual information displayed with the parameters by the Shell.
`Int`	Preset	Like **Text**, but interpreted to be a signed integer
`Multi`	Multi	Allows a single parameter to be displayed as multiple fields. Display using a name for each field, and a custom separator string.
`None`	Common	No other controls are created. Useful for a switch in a list of Parameters
`Parameters`	Parameters	A text field that can allow a sub-selection of more parameters. The sub-parameters are determined using the separator and internal separator strings.
`Password`	Common	A text field with no presets, and Shells should obfuscate its display
`Separator`	Common	Not a parameter, but a visual separator in the Shell application that is displaying the parameters
`Text`	Preset	A text field, with optional presets to fill it in
`TextList`	Common	A list of text items separated by semicolons
`Time`	Preset	Like **Text**, but interpreted to be a time (in milliseconds since midnight, January 1, 1900)
`Uint`	Preset	Like **Text**, but interpreted to be an unsigned integer

Common Parameters

ProcessSequence is derived from RenderJob and includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob. It also adds the following parameters. Note that, unlike the underlying classes or other compiled Modules, the default values for these parameters are defined as part of your Virtual Module definition file. There is no hard coded default for any parameter added by the ProcessSequence Module.

Name	*Type*	*Get*	*Set*	*Meaning*
ActualScene	Text	X		This is a standardized way to access the actual scene file or multiple scene files that are used for a single work operation. If the job is using a single scene file, this is identical to the $(Scene) parameter from RenderJob. However, if the job uses one file per frame, this will expand to the correctly formatted scene file or files that are being worked on in a single work unit. It uses the $(SceneNameFormat) and the $(SubRange) parameters to determine which files are returned.
CheckForMulitpleFiles	Bool			This is a Shell Option that configures whether the shell should try to detect multiple scene source fie sequences.
CommandString	Text	X	X	The string that is used to build the parameters that are passed to the rendering executable to allow it to do its job. Generally, this parameter will contain a string of other variables that will be substituted with other parameters for the work being executed. If set, the Job parameter value will override the Engine Product option. If neither is set, the default comes from the Virtual Module definition file. If no default is given, the work may not execute correctly
EnquoteActual	Bool	X	X	$(ActualScene) can automatically enquote scene files as needed. Because multiple files may be returned and those files are separated by spaces, it can produce a command line syntax error to use the **.Enquote** Parameter Command with that variable. You can use this option to disable the automatic enquoting, if needed.
HideSubrange	Bool	X	X	If Smedge detects that your scene is part of a scene file sequence, setting this will cause the Job variable substitution system to return an empty string for the work SubRange. This allows you to have a command line that will include a frame range only for single scene files.
ImageEndString	Text	X	X	If a line is checked for an image, this text marks the end of the line, and will be trimmed off from the detected filename. You do not need to include any whitespace, because that will be trimmed off automatically.

Name	*Type*	*Get*	*Set*	*Meaning*
ImageNotEndString	Text	X	X	If a line starts with the Image Start String, but ends with this text, it won't be considered a filename. This test is applied before the Image End String test.
ImageStartString	Text	X	X	When parsing the output, lines that start with this text are assumed to contain a rendered image filename. The filename is assumed to start at the first non-whitespace character after this text. If left empty, no filename detection will be performed.
MinimumNumberPadding	Int			This is a Shell Option that configures when searching for a scene file sequence, any numbers found must meet or exceed this many digits to be considered part of a sequence identifier.
SceneNameFormat	Text	X	X	For sequences of scene files, this is the formatting string that is used to create the correct filename for each frame. This value is normally automatically calculated by Smedge and should be blank if the entire scene is in a single scene file
SuccessTexts	Text List	X	X	Allows you to define text that will be searched for at the beginning of a line of output that would signify successful work. If there are no SuccessText entries, then the normal work success tests are done. However, if there are any entries in this section, then there must be at least one line of output that starts with one of the entries, or the work will be considered unsuccessful. You can specify as many of these as you need. Separate multiple entries with a semicolon. The text is not case sensitive.

Reference

The Virtual Module definition file must contain at least a **Module** section. This section contains the basic information that Smedge needs to identify and display the Product to users, and to keep it separate from every other Product. Additionally, you can provide data to create your own custom parameters. Custom parameters will be maintained as part of the Job data, and you can configure how they are interpreted and displayed to users.

The Module Section

This is the information that identifies the virtual Job type in the system. The entries in this section define the Product to the system, and provide many of the defaults for the common parameters that ProcessSequence adds (see Parameters above).

ID

This is a unique identifier for this Job type. Smedge uses the standard UUID (or GUID) format for this value. This is a 16 byte number that can be pretty well guaranteed to be unique in the known universe under most circumstances. You **must** provide an ID that is unique to your whole network, or you will have real problems. ProcessSequence will not allow virtual modules with conflicting IDs to load on a single application, but if you have differences between machines on the same network, you could experience erratic behavior or crash when data from one type of job is forced into a conflicting type.

ProcessSequence expects this number to be in the standard 8-4-4-4-12 hexadecimal format:

```
12345678-9ABC-DEF0-1234-56789ABCDEF0
```

Case does not matter. You can use RLib to generate this value (see the uidgen sample program for an example), or you can use your favorite UUID generating program (for example there is one that comes with the Microsoft Visual Studio compiler).

Command

Virtual Modules work by spawning a child process via a commandline interface. This element provides the actual commandline parameters passed to the executable. Because these modules actually inherit from the ProcessJob base class, you also have a parameter called **Executable**, which can be used to provide the rendering executable program.

What ProcessSequence normally does is build a commandline by getting the value of the **Executable** Job parameter, adding a space if needed, then formatting this command value on the end. But there are many other options. For example, users can override a default **Executable** as an option for the Engine. In this case, if the Job's parameter value is empty, the Engine's option value will be used. If that, too, is empty, then the default set for the type will be used. If there is no default set, no text will be added to the actual commandline.

For the command string part, you can use a variable substitution system to get the actual value of any other Job parameter by name.

You can use any parameter defined in the Virtual Module file, or any parameter defined in the base classes.

Name

This is the nice, user-readable name for this Job type. It is displayed in the Shells for all Jobs of this type.

Shortcut

This is a comma separated list of alternate names that refer to this type. These may be useful as shortcuts for using commandline based Shells to interact with the Job type. For example, the submit program will use these as alternates names to determine what Job type you are trying to create.

Help

This is a short help string that Shells may display to users. In this field (and in this field only) you can use the special character sequence **\n** (a backward slash followed by a lowercase letter N) to signify a line break. If you need to have a literal \n in your string, double the backslash. Because \n is the only sequence that Smedge will look for, you do not need to double the backslashes in any other situation.

Details

A more detailed help string that Shells may display to users.

DefaultEnabled

You can determine if this type will be enabled by default when an Engine encounters it for the very first time. If you set this to a value that will be parsed as true (“True” “Yes” “On” or “1”), then an Engine will enable this Job type the first time it is loaded. Otherwise, the type must be manually enabled by users.

ImageStartText

You can supply one or more values here which the Virtual Module will use to try to detect rendered image filenames from the output. The work unit will watch the output from the process for a line that starts with one of these texts. The first non-whitespace character following whichever string is detected will be assumed to be the beginning of the filename. You can use parameters in this text, and it will be translated when the work starts, based on the current values of the work unit. Separate entries using the semicolon (**;**) character.

ImageEndText

You can supply one or more values that will be search for from the back of any line determined to be an image line using the **ImageStartText** values. If one of these strings is found, the filename will be assumed to end with the last non-whitespace character before this text. Whitespace is automatically trimmed from the parsed lines, so you don't need to add it here if you just want to trim off trailing whitespace. Separate entries using the semicolon (**;**) character.

ImageNotEndText

You can supply one or more value that will be searched for from the back of any line determined to be an image line using the **ImageStartText** values. If one of these strings is found, then any lines that start with the **ImageStartText** value that end with this value will be ignored as potential image filenames. Separate entries using the semicolon (**;**) character.

CheckForMultipleFiles

Virtual Modules can handle products that use either a single scene file for a sequence, or products that use a sequence of scene files, one for each rendered output that is produced. In order to enable the extra processing to detect multiple source scene sequences, set this value to **yes**. The system expects that the last set of digits in the filename are used to put the scene files into sequence. Some examples of sequenced filenames that Smedge will understand:

```
/Volumes/Work/Scene/Folder/Sequence012_ver2.0122.input
X:\Projects\Animation\File01115.input
/usr/data/sequence/0124
```

If you set this value to **no**, or if you do not supply this key in the Virtual Module file, then this processing is disabled, and the scene file will be assumed to be only a single file for all of the frames in the Job.

Custom Parameters

The real power of Virtual Modules comes in defining your own Parameters. You can then use these parameter types as part of your **Command** string and every other place where you need to access custom data. The parameters are accessed using the parameter syntax described above, but are defined in a simple INI format with a few lines of text.

There are actually two sections that you can use to define parameters: **PrimaryParameters** and **SecondaryParameters**. The only difference between the two is the order in which the parameters are added to the Job Type Information. "Primary" parameters are added to the info data before the underlying base classes, and "Secondary" parameters are added after. The order of the parameters has no meaning to the actual operation of the work, but may affect the order in which parameters are displayed in Shells.

The name you supply is the internal name of the parameter, which is the name that is used in parameter substitution. This internal name must not conflict with any other internal name for a parameter or a command in the virtual module, or any of its base classes. See above for a full list of the names already used. Because the parameter name is not generally shown to users (the "NiceName" value is shown instead), it is customary not to use spaces in parameter names. This helps differentiate between parameters and commands. Note that the NiceName, which is displayed to users, can be anything you wish, but it's probably a good idea not to let this name conflict with other names so that you don't confuse users.

The ProcessSequence module will iterate through all of the keys in these sections to determine what parameters are going to be added. The key name is ignored in this iteration, but can be used to determine the order in which parameters will be added, if you care. The iteration will go through the keys alphabetically (case insensitive, ASCII order). Remember this is using a text comparison, so a key of "10" will be parsed before a key of "2".

Each value will be the internal name of a parameter. ProcessSequence will expect that there is a section with a matching name that is used to actually load the parameter information. These sections contain the following elements:

Type

You must provide a type for the parameter. This is used to determine what other data is available in the Job Type Information for this parameter. It is also used by Shells to determine how to format, display or validate the data. See Parameter Types for a list of the types available.

Common Elements

All parameter types can contain these elements.

RealName	This is an optional element that allows you to specify an alternate real (internal) name for the parameter. If no RealName is supplied, the real (internal) name for the parameter will come from the section name in the INI file. This element is useful when you need to ensure that parameter names are case sensitive, because INI files are not case sensitive in Smedge.
NiceName	This is the name of the parameter that will be displayed to the user.
Help	This is some help information that Shells may display to the user when interacting with this parameter.
Default	This is the default value for this parameter type
Flags	This is a comma separated list of the flags that affect how this parameter is used. You can have any combination of the following values:

Name	*Meaning*
`Parameter`	The parameter describes a value member of the Job object

Name	*Meaning*
`Required`	*Modifies **Parameter***. This parameter must have a value associated with it. Without a value associated, the parameter is ignored
`Advanced`	*Modifies **Parameter***. This member is considered an "advanced" parameter. Exact interpretation is left up to the Shell.
`Option`	The parameter describes an option for this Job type.
`Master`	*Modifies **Option***. This option is used by the Master. *Currently ignored…*
`Engine`	*Modifies **Option***. This option is used by the Engine.
`Shell`	*Modifies **Option***. This option is used by the Shells.
`NoOptionDefault`	The default is not used for this parameter when shown as an **Option**.
`NoParamDefault`	The default is not used for this parameter when shows as a **Parameter**.
`NoInputDisplay`	This parameter is not shown by Shells to input or change parameters.
`NoOutputDisplay`	This parameter is not shown by Shells to display Job parameters.

Preset Elements

For the types that can have presets (see the list of types above), all of the common elements are allowed, as well as these elements

Choices	This is a comma separated list of preset choices that can be made available to the user. If items are supplied here, they will be made available as presets in a manner appropriate for the Shell. For example, a GUI may show them as entries in a combo box. But the value is not limited to one of these choices. You must use the **Choice** parameter type if you want that behavior.

Choice Elements

For the **Choice** type, all of the common elements are available, as well as these elements.

Choices	This is a comma separated list of the choices available. These will be the only allowed choices, and even the default must be one of these.

You can have each choice have a value that is displayed to the user that is actually different than the value that is substituted in the parameter substitution. Each choice is in a **DisplayName:ActualValue** format. For example:

```
Choices = Red:r, Green:g, Blue:b
```

This will create a choice that displays "Red," "Green," or "Blue" in a Shell, but that substitutes to **r**, **g**, or **b** in the command string. If you don't put a colon into the choice string, the name and display name will be the same.

You should make sure that the default is one of the choices you have supplied. You must currently use the display name for the choice, not the actual value. If you don’t supply a default, the first choice will be selected by default.

Multi Elements

For the **Multi** type, all of the common elements are available, as well as these elements.

Fields	This is a comma separated list of the fields that make up this multi-value. The count of fields is determined by this list, but empty elements are allowed (i.e., no text between two commas).
Separator	This is the string that is used to break up the value into the fields, and that is inserted between each element when assembling the multiple fields back into a single value. If you don’t supply a value for this, the default, a single space character, will be used.

Bool Elements

For the **Bool** type, all of the common elements are available, as well as these elements

True	This is the text value that will be substituted for a true value. If not supplied, the system will use "Yes" by default.
False	This is the text value that will be substituted for a false value. If not supplied, the system will use "No" by default.

BoolOverride Elements

For the **BoolOverride** type, all of the **Bool** elements are available as well as this element

OverrideText	The text value that is displayed for Job parameters in the third state (when the Engine option provides the actual value for work execution).

Alternate Elements

For the **Alternate** type, all of the common elements are available, as well as this element

Alternate	This is the alternate name that is used. The Name value is used for “true” or “on” and this alternate value is used for “false” or “off”.

Dir Elements

For the **Dir** type, all of the elements of the **Presets** type are available, as well as these elements.

Prompt	This is a prompt string that can be shown to users when they want the Shell to allow them to browse for the directory (or file).

File Elements

For the **File** type, all of the elements of the **Dir** type are available as well as these elements

Filter	This is a filter string that can be used by the file selection dialog to make it easier to find files based on their extensions. The filter part will be a file selection filter like "C++ Files (*.cpp, *.cxx)|*.cpp;*.cxx|{All Files}". No filter will default to {All Files}. If the string includes the single filter "{All Files}" (case insensitive and including the {}s) the appropriate actual filter should be added by Shells for the platform they are running on. If the string includes the single filter "{Executable Files}" (case insensitive and including the {}s) the appropriate actual filter should be added by Shells for the platform they are running on.

Parameters Elements

For the **Parameters** type, all of the common elements are available, as well as these elements.

InternalSeparator	This is the string that is inserted in between a child parameter name (real name) and its value if that child parameter is being added to the full parameter string. Defaults to a single space if not supplied in a parameter definition.
Parameters	This is a comma separated list of the parameter names that make up this list of parameters. This allows you to create a group of parameters accessed with a single parameter name. SmedgeGui displays this parameter type as a separate tab in the Submit Job window. (See the User Manual for more information about this feature.)
Separator	This string is inserted between the name (real name) of the parameter and its child parameters string. Defaults to a single space if not supplied in a parameter definition.

First you give each child parameter the name of the actual commandline switch that it uses. Then you create these parameters to work in a manner that allows users to easily set and select which extra parameters they want to add to their commandline. The easiest way to understand this is to see it in action. Check out the **Extras** parameter in the following example PXS file, and check out how it works in the SmedgeGui Shell application.

Custom Commands

As with parameters, you can add custom commands to your Virtual Module. Currently, these commands are limited to executing a commandline (after performing parameter substitution on it, of course!). Just like with the parameters, you can add your commands in either the **PrimaryCommands** section or the **SecondaryCommands** section. The only difference is that the "Primary" commands are added before commands from the base classes, and "Secondary" commands are added after. The order of the commands is not important to Smedge, but may affect the order they are displayed in Shells.

Each command must have a name that does not conflict with any parameter or command either defined in the Virtual Module file or from any base class. Because Commands names are usually displayed directly to the user, the convention is to use spaces in command names. Unlike Parameters, commands do not have a separate display name.

Command Elements

Commands have three elements.

Command	This is the actual commandline that will be substituted and executed when this command is triggered by a user.
Help	This is a string that Shells can display to let the user know what this command is going to do
RestrictionName	This is a string that Shells can use as part of the voluntary restriction system in Smedge. If you give your command a restriction name, then you can use the Configure Master dialog box to set a restriction on that command or not, using your custom name. Any restricted command will be unavailable at run time if the RestrictionName is set as restricted, unless the application is running as an administrator.
Flags	This is a comma separated list of flags, which can be any combination of the following values:

`ForParent`	This command applies to parent Jobs
`ForChild`	This command applies to child Workers
`Separator`	A placeholder to insert a separator between commands. Shells should interpret this in a manner appropriate to how that Shell displays commands to users.

Other sections

ErrorText

This section allows you to define text that will be searched for at the beginning of a line of output that would signify an error in the work unit. Smedge will then assume that the work unit has failed and will requeue it.

You can specify as many of these as you need. The key name does not matter, but must be unique for each string you want to search for. The text search is case sensitive.

SuccessText

This section allows you to define text that will be searched for at the beginning of a line of output that would signify successful work. This section will take precedence over any other error detection systems. If there are entries in this section, then there must be at least one line of output that starts with one of the entries, or the work will always be considered unsuccessful. However, if there are no entries in this section or if this section is missing from your Virtual Module file, then the normal error detection systems will be used.

You can specify as many of these as you need. The key name does not matter, but must be unique for each string you want to search for. The text search is case sensitive.

OverrideDefaults

This section allows you to override the default values for any parameter. Since you can just provide the default for custom parameters defined in the file, this really only makes sense to override defaults for parameters from the base classes. Use the parameter name as the key and supply your new default value.

Note that parameter defaults will still follow the defined flag for allowing display of the default value (the **NoOptionDefault** and **NoParamDefault** flags are not modified). See Custom Parameters for more information.

OverrideFilters

This section allows you to customize the filter strings for any File type parameters defined in ProcessSequence or any of its base classes. The key should contain the internal name of the File parameter you want to modify the filter for, and the value specifies the new filter string value.

OverrideFlags

This section allows you to customize the flags for any parameter defined in ProcessSequence or any of its base classes. The key should contain the internal name of the parameter you want to modify the flags for, and the value specifies a comma separated list of the flags you want for that parameter. The flags are replaced with the flags you specify, so be sure to specify all of the flags that the parameter will require or you may find that things don't work as you expect them to. See Parameter Flags for a list of all the flags available and what they mean.

AutoDetect

This section allows you to configure a basic auto-detect ability to set the value of one or more other parameters when a parameter value is changed by a user in a Shell. This system is not as sophisticated as the functionality you would have if you used a compiled module, because compiled modules give you full access to the Smedge API and whatever 3rd party libraries you link into your module. However, it does give a basic functionality using the variable command syntax described in the Command section.

The key is the name of the parameter that triggers the auto-detect change. When this parameter is changed by the user, the parameters listed in the value will then be changed as well, using the current values of any parameters requested. The basic syntax is ***Value* = *Change***[, *Change…*]. Each ***Change*** is in a ***ChangeValue* = *ChangeTo*** format. For the changes, the **ChangeValue** is the name of the parameter that you want to automatically modify. No variable substitution is performed on this string. The **ChangeTo** string will be variable substituted.

This syntax can be somewhat complicated. It is also additionally complicated by the use of the = character in the Value, which means that you should not break these lines up onto multiple lines in your Virtual Module file. Examine this sample:

```
Scene = Name = [$(Project): ]$(Scene.File)
```

The *Value* that triggers the auto-detect is **Scene**. When the **Scene** parameter is changed by the user, the *Change* that will be executed is "Name = [$(Project):]$(Scene.File)". This auto-detect has only one Change. The *ChangeValue* is **Name**, which means that when the **Scene** parameter is modified, the **Name** parameter will then also be automatically modified with the *ChangeValue* of "[$(Project):]$(Scene.File)". The current value of the **Project** parameter and the new value of the **Scene** parameter will be run through the variable substitution system to calculate the actual value that will then be assigned to the **Name** parameter.

Example File

This is a simple example Virtual Module file that shows the basic structure and syntax. It does not actually support any particular renderer, but is useful as an example of creating the file. Note that any comments (in green and starting with a ; at the beginning of the line) are totally optional, and whitespace in and around the section and key strings is ignored. See the RLib INI syntax reference for more information.

Also be sure to look at the Virtual Module files that are included in the Smedge distribution to see other examples of what you can do with a Virtual Module, and how to accomplish it.

```
; ProcessSequence.psx

;
; This is a sample virtual Module file for a ProcessSequence.sx Module.
;
; Smedge 3
; Copyright (c) 2004-2005 Überware

; The Module section includes the basic module information

        [ Module ]

; Every Job type in Smedge needs its own unique ID. Make sure to modify this
; ID for each new type. Note that Jobs and options will store this type as
; well, so if you change this ID, you will lose any saved options, and be
; unable to use any saved jobs that rely on it. You can use the included
; commandline too uidgen.exe to create new IDs, or any other tool that will
; create a Globally Unique ID.

ID                  = 46D1CA19-FD89-4ad5-9A43-676A28764C3D

; The actual commandline is generated by assembling the executable and command
; fields and doing a member name substitution on any special parameters
; marked. Parameters are marked with the notation $(name). Any parameters from
; ProcessSequenceJob or its base classes is available for use, as well as any
; custom parameters defined in this file. For a complete list of parameters
; available, see the documentation for the Job derived classes: SequenceJob,
; ProcessJob, and RenderJob (more may be available in later versions of Smedge).
;
; The most common parameters you will use will be:
;
; $(Scene)          - The scene the user can specify for the Job
; $(SubRange)       - The full range of the work formatted into a x-y string (no spaces)
; $(SubRange.Start) - Just the start of the range of the work
; $(SubRange.End)   - Just the end of the range of the work
; $(CPUs)           - The number of CPUs that are assigned to this job.
;
; If you want to have a portion of the commandline conditional on a non-empty
; parameter value, you can surround that portion with square brackets: []
;
; The Command gives the defaults for these values.
; All values are overridable at both the Job and Engine levels.

Command             = [-proj $(Project) ]-start $(SubRange.Start) -end $(SubRange.End)
                      -format $(Format) [$(Extra) ]$(Scene)
```

```
; Because ProcessSequence is derived from the RenderJob base class, it
; automatically gets all of the functionality of a Rendering app, including
; automatic frame name detection and verification. To enable this, you can
; provide output parsing strings here for detecting the filenames from the
; output of the spawned process. If the detected names are not absolute
; paths, the Engine will look for a parameter called ImageDir and prepend
; that value, or, if that value is empty or cannot be found, the
; StartDirectory value will be prepended.

; Look for lines that with this text. The first non-white spcace text
; following this text is assumed to be the start of the filename. You can use
; parameters in these fields, and they will be translated when the Work is
; started


ImageStartText      = Rendering:

; Trim this text from the end of the line. Whitespace will be trimmed
; autmoatically, so you don't need to add it here.

ImageEndText        = .


; You can specify text that indicates that lines which may start with the
; ImageStartText string may not actually be images. If the line ends with
; this text, it will not be processed as an image. This test is done before
; the ImageEndText is checked and removed.

ImageNotEndText     =

; The rest of the fields allow you to customize how users interact with your
; custom type

; This is the name displayed to the user for this type of Job

Name                = My Job Type

; This is a shortcut you can use to create jobs of this type, say by a
; commandline. Values can be separated by commas. Whitespace surrounding values
; will be ignored.

Shortcut            = myjobtype, myjob

; This is a short help string that some Shells may display to the user about this type

Help                = This is a custom ProcessSequenceJob type

; This is a more extended help text that some Shells may display to the user about this type

Details             = There are no Shells that currently display this information, but
                      we're working on that. In the mean time, here's a nice long string of
                      information to get you started.

; This allows you to determine if this type should be enabled by default on
; an Engine that has never seen it before.

DefaultEnabled      = No



; The PrimaryParameters section allows you to add custom parameter for the
; type. These parameters are added before any parameters from the underlying
; base classes, so they will generally appear above them in Shells. The order
; of parameters is only important to Shells. You can order your parameters by
; using a number for the key name. If two parameters have the same number, the
; last one read will be the one used.
```

```
;
; Parameters must have a unique name. This name is used internally to access
; the parameter value, and is used as a section name in this file to describe
; all of the parameter information details. The parameter name must not
; conflict with any other parameter or command, either in this virtual Module
; file or in any of the underlying base classes. The parameter name is not
; generally shown to users, so the general convention is to not use spaces in
; parameter names, and use spaces in command names.


    [ PrimaryParameters ]


1                   = Format


; The SecondaryParameters section is the same as the PrimaryParameters, except
; that these parameters are added to the Job type information after any
; parameters from the underlying base classes. This means that they will
; appear below the underlying parameters in Shells. The order of parameters is
; only important to Shells. The names of the parameters must not conflict with
; any other parameters or commands, either in this file or in any of the
; underlying base classes. The parameter name is not generally shown to users,
; so the general convention is to not use spaces in parameter names, and use
; spaces in command names.


    [ SecondaryParameters ]

1                   = Extra
2                   = Project


; The PrimaryCommands section allows you to add custom commands to the type.
; Currently commands are limited to executing an external program via a
; commandline. As with the parameters, Primary commands are added to the list
; before the underlying job classes. The only affect this has is the order of
; display of commands in Shells. Commands can be ordered by a number in the key
; name. The Command name must not conflict with any other parameters or
; commands. A command's name is also what is displayed to a user, so the
; general convention is to use spaces in Command names an no spaces in
; Parameter names.


    [ PrimaryCommands ]

1                   = View Format Info
2                   = Separator


; The SecondaryCommands is just like the PrimaryCommands, except that these
; commands are added after the underlying Job base classes.


    [ SecondaryCommands ]

; The ErrorText section allows you to define text that will be searched for
; at the beginning of a line of output that would signify an error that Smedge
; should assume would signify a failed work unit. You can specify as many of
; these as you need, and the key name does not matter (but must be unique for
; each different value you need). The text is case sensitive.

    [ ErrorText ]

1                   = ERROR:
```

```
; The SuccessText section allows you to define text that will be searched for
; at the beginning of a line of output that would signify successful work. If
; there are no SuccessText entries, then the normal work success tests are
; done. However, if there are any entries in this section, then there must be
; at least one line of output that starts with one of the entries, or the work
; will be considered unsuccessful. You can specify as many of these as you need
; and the key name does not matter (but must be unique for each different value
; you need). The text is case sensitive.

    [ SuccessText ]


; The OverrideDefaults section allows you to override the default values for
; any parameter. Since you can just provide defaults for custom parameter types
; in their definition, this really only makes sense to override defaults for
; parameters from the base classes. Use the parameter name as the key and
; provide your new default.
;
; Note that parameter defaults will still follow the defined flag for allowing
; display of a default for a Job parameter or an Engine/Shell option. See the
; documentation of ParameterInfo for more information.


    [ OverrideDefaults ]

Executable          = MyProgram
StartDirectory      = $(Project)/images


; The AutoDetect section allows your virtual module to perform a basic
; auto-detect ability to set the value of one or more other parameters
; of a job based on the values of other parameters. This is not as cool as
; what you get for doing a compiled module, where you have full access to the
; APIs for both Smedge and whatever 3rd party libraries you want to link to
; your module (e.g., the Maya module, if you wanted to actually open and work
; with Maya files in the AutoDetect logic.)
;
; The keys are in a Value = change[, change ...] sequence. The Value is the
; parameter that has just been modified. Each change in the list is also in
; a value=change syntax. The value here is the name of the parameter you
; want to modify, and the change is a string that will be run through the
; FormatStringWithParameters function, like the commandline. Use $(Name)
; syntax to access other job values

    [ AutoDetect ]

Scene               = Name = [$(Project): ]$(Scene.File)

; Now we have a section for each parameter and command to actually define the
; functionality. The name of these sections must correspond tothe names used
; above.


;
;                 Parameters
;


    [ Project ]


; This is an example of a Parameter definition. You must provide at least a
; type and a name, and a flag if this is a Job object parameter or an option
; for the type. See the documentation for ParameterInfo for full details
; and requirements.
```

```
NiceName            = Project
Type                = Dir
Help                = This is the project base directory
Default             =
Flags               = Parameter

; This is specific to the Dir type:

Prompt              = Select the base project directory


     [ Format ]

NiceName            = Scene Format
Type                = Text
Help                = This is the custom type format. You must provide a valid format, or select one from the list
Default             = ascii
Flags               = Parameter, Required

; This is specific to the Text type (technically to all PresetsParameterInfo types):

Choices             = ascii, binary


    [ Extra ]

NiceName            = Extra Parameters
Type                = Parameters
Help                = Extra commandline parameters you can pass to the executable.
Flags               = Parameter, Advanced

; This is specific to the Parameters type:
; These are all of the sub-parameters that are used to generate this value

Parameters          = -im, -x, -alpha, X


    [ -im ]


NiceName            = Image Name
Type                = Text
Help                = The base filename for the images
Flags               = Required


    [ -x ]


NiceName            = Resolution
Type                = Multi
Help                = The X and Y resolution in pixels
Flags               = Required

; These are specific to the Multi type:
Fields              = X, Y
; Notice the whitespace is made part of the separator by using quotes. The
; quote marks will be removed when the file is read, but the spaces inside
; will be left as part of the value of Separator.
Separator           = " -y "


    [ -alpha ]


NiceName            = Render Alpha
```

```
Type                = Bool
Help                = Include the alpha channel in the rendered image file
Flags               = Required


    [ X ]

RealName            = -x
NiceName            = Capital X
Type                = Bool
Help                = This parameter should appear as a capital X separate from the lower case x for the Resolution
Flags               = Required

;
;                Commands
;


    [ View Format Info ]

; This is an example of a command section

Command             = $(Executable) -viewFormatInfo $(Format)
Help                = Try to view the completed frame from the renderer
Flags               = ForChild, ForParent


    [ Separator ]

Flags               = Separator
```

Product Reference

This is a reference for all Products that currently ship with Smedge. The following information provides the Smedge default Products from the suite of Job Modules included with Smedge. Many of these Modules support dynamic creation of additional Products, and your system may include different or additional Products. See the chapters on Dynamic Products, Maya Products, and Virtual Modules, or talk to your system administrator for more information about custom or third party Modules you may have installed.

3D Studio Max

This Module is designed to control the 3ds max command line renderer included in versions 6.0 and later. This is a Compiled Module. It includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob. The Max.sx Module can be customized to support multiple versions of Max with the Dynamic Products configuration system, with one extension: The "SplitFrame" key is a Boolean value that determines if the customized Product is a sequence renderer or a single frame renderer (see 3D Studio Max (SingleFrame))

Copying the scene locally

Max has some issues when used in larger networks. Autodesk has confirmed that the Max program code itself can cause failures reading the scene file when a large number of nodes try to read the same file simultaneously. To work around this problem, the Max.sx Module has a system to copy the scene file to the local Engine's drive before starting the max renderer. In so doing, it can also generate a path file with all of the directories and subdirectories of from the directory in which the original scene file is located. This way, any textures can be referenced relative to the scene file and can still be accessed even if the scene file is copied to a different folder. If you have specified a path file for the job, the paths from that file are also read and added to the generated path file.

This system is implemented internally in the Module. You can use the **CopyLocally** parameter of a Job or option for an Engine to enable or disable this behavior. All Max Products (both single and multi-frame) can use this system. By default, it is not enabled. See the Job Parameter Command CopyLocally for more information about the local copy system that is used to implement this behavior.

General Information

Default ID	`A4F726B0-A613-4f2b-94FB-42C812857459`

Type Name and Shortcuts	`3D Studio Max`
	`Max`
	`Max6`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
CopyLocally	BoolOverride	X	X	Enables the Max scene file to be copied locally. See Copying the scene locally.	Engine Default (no)

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
LocalPathFile	BoolOverride	X	X	Enables the generation of the local path file (the MXP file) when the copy locally system is enabled. The local MXP file will not be generated if this is not on, and the renderer will only look for textures in the default locations and any locations specified in the original MXP file.	Engine Default (yes)
NthFrame	Text	X	X	Allows you to change the number of frames rendered in a sequence. Make sure that your Packet Size is set to a whole number multiple of this value. If blank, it will default to 1	
PathFile	File	X	X	A path configuration file (.mxp)	
TimeLimits	Parameters	X	X	Max time limits	
WorkPath	Dir	X	X	Root location for job data folders	

3D Studio Max (Single Frame)

This Module is designed to control the 3ds max command line renderer included in versions 6.0 and later to split a single frame across multiple machines. This is a Compiled Module. It includes all of the parameters from Job, ProcessJob, SliceDistributor, and RenderJob. The Max.sx Module can be customized to support multiple versions of Max with the Dynamic Products configuration system, with one extension: The "SplitFrame" key is a Boolean value that determines if the customized Product is a sequence renderer or a single frame renderer (see 3D Studio Max)

Single Frame Max Products can also make use of the system for Copying the scene locally.

General Information

Default ID	`e59d00dd-3c2e-4994-b665-ca5fdcbc92ed`

Type Name and Shortcuts	`3D Studio Max (Single Frame)`
	`Max single`
	`Max1`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
CopyLocally	BoolOverride	X	X	Enables the Max scene file to be copied locally. See Copying the scene locally.	
Extra	Parameters	X	X	Extra parameters you want to send on the command line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Overlap	Int	X	X	The amount of overlapping pixels that border each strip	10
PathFile	File	X	X	A path configuration file (.mxp)	
TimeLimits	Parameters	X	X	Max time limits	
WorkPath	Dir	X	X	Root location for job data folders	

3D Studio Max (via SFRender)

This Module is designed to control 3ds max via the SFRender third party commandline interface. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`42906ff2-25b4-4021-a391-39238e96922f`

Type Name and Shortcuts	`3D Studio Max (via SFRender)`
	`max3`
	`max4`
	`max5`
	`sfrender`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	

3Delight

This Module is designed to control 3Delight renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`7582c8ad-cafd-4ac2-b5f0-a20101872527`
Type Name and Shortcuts	`3Delight`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	

After Effects

This Module is designed to control the AfterEffects 6.x or later commandline renderer. This is a Compiled Module. It includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

This Module supports the creation of Dynamic Products with a simple INI file format. See the Dynamic Products chapter for more information.

Using Smedge with After Effects

After Effects works differently than most other rendering systems. Because of the variety of ways that you can configure and use AE to render, see the supplemental *Using Smedge with After Effects* manual included in the documentation folder of the Smedge distribution.

Note that you cannot currently use Smedge to control After Effects rendering to a single movie file. You will need to render to an image sequence of some kind, which you can then convert to a movie file if you need.

General Information

Default ID	`ef7f0373-3542-4d1d-80d0-bb8599fa4c63`

Type Name and Shortcuts	`After Effects`
	`AE`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Comp	Text	X	X	The name of the comp in the scene file to render	
Extra	Text	X	X	Extra commandline parameters you wish to pass to the renderer.	

Air

The **Air** Product allows you control the Air renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`1ee7351d-6d52-48f7-9bb8-df232d2faca4`
Type Name and Shortcuts	`Air`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	

Aqsis

The **Aqsis** Product allows you control the Aqsis command line renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`89f86e8d-578e-4859-96b0-102153b79a32`
Type Name and Shortcuts	`Aqsis`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Crop	Multi	X	X	Define a crop window. Only the portion of the image inside the specified region will be rendered. The co-ordinates are in screen space, so a value of 0.0 is at the top resp. left and a value of 1.0 is at the right resp. bottom, irrespective of the actual resolution. Using this option is equivalent to the RIB command Crop-Window x1 x2 y1 y2.	
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Res	Multi	X	X	Set the output image pixel resolution regardless of what is specified in the RIB.	

Alias

This Module is designed to control the Alias Studio commandline renderers. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`374e7b28-645c-4385-a3Cb-c5fc7f45ac11`

Type Name and Shortcuts	`Alias Studio`
	`Alias`
	`Studio`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Product	Choice	X	X	Which Alias renderer to use for the Job. Your choices are: Renderer RayTracer PowerCaster PowerTracer	PowerTracer

Blender

This Module is designed to control the Blender commandline renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`75316b6c-2510-44f1-b15d-d6a8f23a4c3f`
Type Name and Shortcuts	`Blender`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	

Cinema 4D

This Module is designed to control the Cinema 4D commandline renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`986184cd-04c1-451f-af66-f2947e405434`

Type Name and Shortcuts	`Cinema 4D`
	`C4d`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	

Combustion 4

This Module is designed to control the Combustion commandline renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`0dcfc181-7272-4d27-8042-cf208b8f05b5`

Type Name and Shortcuts	`Combustion 4`
	`C4`
	`Combustion`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
RenumberBy	Text	X	X	The increment value for renumbering the frame files. If blank, defaults to 1	
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-b** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

Digital Fusion

This Module is designed to control the Alias Studio commandline renderers. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`aefec858-eeb1-4e67-8b75-d147e0b60a0b`

Type Name and Shortcuts	`Digital Fusion`
	`Fusion`
	`DF`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
rtPath	File	X	X	The name (and optionally the path to) the dfscript that actually controls the rendering.	Rendertool
dfPath	File	X	X	You can supply the full path to the Digital Fusion executable to the script. If you leave this blank, the default supplied script will check the environment variable FUSION_LOCATION, then the start directory for a copy of Digital Fusion or the Render Node.	

finalRender for Maya

The **Maya.sx** compiled Module now includes the ability to set up customized products for different renderers accessible through the Maya command line interface. Each Product is a section in the Maya.ini file which is in the Module folder. You can add your own or customize the existing ones in a similar manner to how you create Virtual Modules.

The **finalRender for Maya** Product allows you to tell Maya to render using the Maya Vector renderer. This will override any Render Layer renderer settings in your Maya 7 file.

The finalRender for Maya Product includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

General Information

Default ID	`a9bdf783-810b-46ce-885e-90d37f2e4128`

Type Name and Shortcuts	`finalRender for Maya`
	`fr4m`
	`fr`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Project	Dir	X	X	The Maya project to use for rendering. I left blank, no project will be passed to the renderer.	
RenderDir	Dir	X	X	Override the output folder	
RenumberBy	Text	X	X	The increment value for renumbering the frame files. Blank defaults to 1	
RenumberStart	Text	X	X	The Job can be renumbered starting at this value. If left blank, the file frame numbers will correspond to the frames rendered	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-b** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

Fryrender

This Module is designed to control the Maxwell Light Simulator command line renderer. This is a Compiled Module. It includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

Fryrender/dsimerge or Frycmd

The Fry.sx Module can handle the command line operation for both Fryrender using dsimerge to merge the DSI files together (version 1.0), and for Frycmd, which can also be used to merge DSI files (version 1.5). If you set the command line to the Fryrender executable, the command line will automatically be created with the extra flags required, and DSI merging will use the dsimerge command. If you set it to the Frycmd executable, that executable will be used for both the rendering and merging DSI files.

DSI Generation

Fry can generate a proprietary output format that it can use to “merge” the results from several machines into a single file, improving quality. Because these files are so large, Smedge always creates them locally on the machine during the render process. Once finished, Smedge intelligently handles the transfer and merging of these files when you choose to repeat the job.

You can optionally request the Smedge use Large File Transfer jobs to queue the moving, instead of having the move happen immediately as part of the work process. This can decrease the load on your network as these very large files are moved around between machines, but may slow down the merge process depending on your settings for distributing Large File Transfer work.

General Information

Default ID	`26BD6DFF-1F01-4CD3-A326-94722F132039`
Type Name and Shortcuts	`Fryrender` `Fry`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
DistributionMode	Choice	X	X	Determines how repetitions of sequences are handled. Set to “0” to dispatch every repetition of a frame before moving to the next frame. Set to “1” to have every frame distributed once before the successive repetitions of each frame.	1

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
DSI	File	X	X	Specify the DSI file path used during rendering	
Output	File	X	X	Specifies the full path and name of the image file. The file name can refer to any of the multiple graphic formats supported.	

Custom Commands

Fry also adds some additional commands that you can use to control the rendering process.

Stop and Merge	This will stop a current running render, and allow it to be merged immediately at its current quality level. Note that this only affects the specific work unit(s) you have selected when you request the command, and won't stop any additional renders from starting in order to finish all of the requested repetitions.
Stop All Work and Merge	This will stop all currently running renders and start the merge process for all of them. Additionally, no further renders will be started, even if there are outstanding repetitions that haven't been rendered yet. You can access this command by selecting any work unit from the Job, the Job itself, or any history element from the job.

Gelato

This Module is designed to control the Gelato renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`0e304886-b40b-4471-8e08-3f431442c3e1`
Type Name and Shortcuts	`Gelato`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	

Generic Script

This Module is designed to allow the distribution of any commandline. You can use it to distribute a range based sequence, such as a rendered animation sequence, or you can use it to distribute a command to execute exactly one time on every Engine that is a member of the assigned Pool. If you supply a **Range** value, the Master will use that range to break up the distribution just like every other range based Product. However, if you leave the **Range** blank, the work will be evenly distributed to every Engine in the pool. When you submit a Generic Script Job with **Submit**, you will still need to supply a –Range flag to the **Submit** commandline, but you can leave its value blank. **This is different than other Products.**

Generic Script is a Compiled Module. It includes all of the parameters from Job, ProcessJob, and SequenceDistributor.

General Information

Default ID	`2c0ad30d-5432-44f3-8ab9-5d09d08e2955`

Type Name and Shortcuts	`Generic Script`
	`Generic`
	`Script`

Parameters

Name	*Type*			*Meaning*	*Default*
Command	Text	X	X	The command line to execute. The Engine will perform standard Smedge style variable substitution before launching the command. You can access other Job parameters using the **$(***Name***)** syntax.	

Houdini

This Module is designed to control the Houdini command line renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`044da801-d94b-41bd-9861-ec429c7ee6e4`
Type Name and Shortcuts	`Houdini` `Mantra`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
PythonFilter	File	X	X	This is the path to a Python script file that is used to do filtering of the Houdini IFD Files before rendering.	

Large File Transfer

This Module allows you to queue file operations in order to reduce your network load. File operations are queued separate from other Smedge workers, but can still be limited in the number of simultaneous operations allowed. This is a Compiled Module. It includes all of the parameters from Job.

General Information

Default ID	`3e732d37-a865-45f3-b2b2-3624db07ce2d`

Type Name and Shortcuts	`Large File Transfer`
	`lft`
	`copy`
	`move`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Operation	Choice	X	X	The type of operation to perform. Choices are: 0 = Move/Rename a file 1 = Copy a file	1
Overwrite	Bool	X	X	Allow the target file to be overwritten if it exists	Yes
Source	File	X	X	The source file	
Target	File	X	X	The target file	

Lightwave

This Module is designed to control the Lightwave command line renderer. This is a Compiled Module. It includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

This Module supports the creation of Dynamic Products with a simple INI file format. See the Dynamic Products chapter for more information.

General Information

Default ID	`0c09288e-7241-439c-af2b-b9954c61fb6d`

Type Name and Shortcuts	`Lightwave`
	`LW`
	`LWSN`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
ConfigFile	Dir	X	X	The full path to the Lightwave config files to use for rendering.	
ContentDir	Dir	X	X	The path to the content directory to use for rendering.	
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence. Make sure that your **PacketSize** is set to a whole number multiple of this value. If blank, defaults to 1	

Maxwell Light Simulator

This Module is designed to control the Maxwell Light Simulator command line renderer. This is a Compiled Module. It includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

Maxwell 1 and Maxwell 2

The command line syntax and functionality of the rendering has changed significantly in various versions of Maxwell. To support this, Smedge now can set a Maxwell job to use a command line formatted correctly for Maxwell 1.6.0 and earlier (with no output), Maxwell 1.6.1 – 1.7 (with output), or Maxwell 2.0 or later (with a revised and extended syntax). Make sure you choose the correct syntax version for the version of Maxwell you are using for rendering. See the **CommandFormat** parameter for more information.

MXI Generation

Maxwell can generate a proprietary output format that it can use to "merge" the results from several machines into a single file, improving quality. Because these files are so large, Smedge always creates them locally on the machine during the render process. Once finished, Smedge intelligently handles the transfer and merging of these files when you choose to repeat the job.

You can optionally request the Smedge use Large File Transfer jobs to queue the moving, instead of having the move happen immediately as part of the work process. This can decrease the load on your network as these very large files are moved around between machines, but may slow down the merge process depending on your settings for distributing Large File Transfer work.

To properly allow MXI merging, you need to configure the path to the mximerge executable as well as the path to the normal maxwell render command. You can set this path, along with the options for using the Large File Transfer job as Engine Product options. See the User Manual for more information on setting Product Options in Smedge.

General Information

Default ID	`644d0701-8027-48e0-8bf8-ea8851a519f2`

Type Name and Shortcuts	`Maxwell Light Simulator`
	`Maxwell`
	`mx`
	`mxcl`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
CommandFormat	Choice	X	X	Sets the command line format for the version of Maxwell that you are using. Choices: 0: version 1.0 – 1.6.0 1: version 1.6.1 – 1.7 2: version 2.0 +	2
Display	Choice	X	X	Select the options for displaying the Maxwell program while a render is running. (Note that you cannot display an interface when SmedgeEngine is running as a background Service). 0: No display 1: Console display only 2: Full GUI display	0
DistributionMode	Choice	X	X	Determines how repetitions of sequences are handled. Set to “0” to dispatch every repetition of a frame before moving to the next frame. Set to “1” to have every frame distributed once before the successive repetitions of each frame.	1
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.) Note that there are some extra parameters that are only allowed for Maxwell 2.0+. Smedge does not verify that you have specified flags in this parameter that are allowed for the command format you are using. Be sure to check that you are not using flags that are not allowed, or you will have work units that fail.	
MXI	File	X	X	Specify the MXI file path used during rendering	
Output	File	X	X	Specifies the full path and name of the image file. The file name can refer to any of the multiple graphic formats supported. In case of sequences, the output files will be numbered with a 4-digit suffix.	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
UseZeroCPUs	Bool-Override	X	X	When Smedge starts a worker on a machine that is to use all of the Engine's CPUs, you can tell Maxwell to use the number 0 instead of the actual CPU count. This can lead to some performance gain in Maxwell.	Engine Default (yes)
Verbose	Choice	X	X	Sets the verbosity level from the renderer (Maxwell 2.0+) 0: None 1: Errors 2: Warnings 3: Info 4: All	3

Custom Commands

Maxwell also adds some additional commands that you can use to control the rendering process.

Stop and Merge	This will stop a current running render, and allow it to be merged immediately at its current quality level. Note that this only affects the specific work unit(s) you have selected when you request the command, and won't stop any additional renders from starting in order to finish all of the requested repetitions.
Stop All Work and Merge	This will stop all currently running renders and start the merge process for all of them. Additionally, no further renders will be started, even if there are outstanding repetitions that haven't been rendered yet. You can access this command by selecting any work unit from the Job, the Job itself, or any history element from the job.

Maya

The **Maya.sx** compiled Module now includes the ability to set up customized products for different renderers accessible through the Maya command line interface. Each Product is a section in the Maya.ini file which is in the Module folder. You can add your own or customize the existing ones in a similar manner to how you create Virtual Modules.

The default Maya product is designed for use with Maya 7.0 and later. It can handle the extended Render Layers feature new in Maya 7, which allows you to assign different renderers for each layer. The downside of this new system is that you have less control of the render settings via the Extra Parameters you can supply to the command line. If you are using Maya 6, or if you want more command line controls, you can use the **Maya Software** Product.

The Maya Product includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

General Information

Default ID	`56f3b0da-a949-4c76-a21e-5bff03aca8af`
Type Name and Shortcuts	`Maya` `file`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Project	Dir	X	X	The Maya project to use for rendering. I left blank, no project will be passed to the renderer.	
RenderDir	Dir	X	X	Override the output folder	
RenumberBy	Text	X	X	The increment value for renumbering the frame files. Blank defaults to 1	
RenumberStart	Text	X	X	The Job can be renumbered starting at this value. If left blank, the file frame numbers will correspond to the frames rendered	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-b** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

Maya (1-5)

The **Maya.sx** compiled Module now includes the ability to set up customized products for different renderers accessible through the Maya command line interface. Each Product is a section in the Maya.ini file which is in the Module folder. You can add your own or customize the existing ones in a similar manner to how you create Virtual Modules.

The Maya (1-5) Product allows you to control older versions of Maya using the old command line syntax.

The Maya (1-5) Product includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

General Information

Default ID	`8f6790c1-7ea9-4091-82b9-e7c64327a632`

Type Name and Shortcuts	`Maya (1-5)`
	`Maya1`
	`Maya2`
	`Maya3`
	`Maya4`
	`Maya5`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Project	Dir	X	X	The Maya project to use for rendering. I left blank, no project will be passed to the renderer.	
RenderDir	Dir	X	X	Override the output folder	
RenumberBy	Text	X	X	The increment value for renumbering the frame files. Blank defaults to 1	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
RenumberStart	Text	X	X	The Job can be renumbered starting at this value. If left blank, the file frame numbers will correspond to the frames rendered	
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-b** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

Maya Hardware Renderer

The **Maya.sx** compiled Module now includes the ability to set up customized products for different renderers accessible through the Maya command line interface. Each Product is a section in the Maya.ini file which is in the Module folder. You can add your own or customize the existing ones in a similar manner to how you create Virtual Modules.

The **Maya Hardware** Product allows you to tell Maya to render using the Maya Hardware renderer. Note that hardware rendering is actually dependent upon the hardware installed on your Engines. Using this product will override any Render Layer renderer settings in your Maya 7 file.

The Maya Hardware Renderer Product includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

General Information

Default ID	`bbf04770-cb4b-40b7-8d2d-ebee98c3a767`

Type Name and Shortcuts	`Maya Hardware Renderer`
	`Maya Hardware`
	`hw`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Project	Dir	X	X	The Maya project to use for rendering. I left blank, no project will be passed to the renderer.	
RenderDir	Dir	X	X	Override the output folder	
RenumberBy	Text	X	X	The increment value for renumbering the frame files. Blank defaults to 1	
RenumberStart	Text	X	X	The Job can be renumbered starting at this value. If left blank, the file frame numbers will correspond to the frames rendered	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-b** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

Maya Lightmap Generator

The **Maya.sx** compiled Module now includes the ability to set up customized products for different renderers accessible through the Maya command line interface. Each Product is a section in the Maya.ini file which is in the Module folder. You can add your own or customize the existing ones in a similar manner to how you create Virtual Modules.

The **Maya Lightmap Generator** Product allows you to tell Maya to render using the metnal ray lightmap renderer. Using this product will override any Render Layer renderer settings in your Maya 7 file.

The Maya Product includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob. **Note:** Currently the Maya lightmap renderer does not support overriding the frame range to render. You can only send this type of Job as a single whole work unit for the entire scene.

General Information

Default ID	`cb4dcb75-6330-47df-af47-cae6fe573389`

Type Name and Shortcuts	`Maya Lightmap Generator`
	`Lightmap`
	`lm`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Project	Dir	X	X	The Maya project to use for rendering. I left blank, no project will be passed to the renderer.	
RenderDir	Dir	X	X	Override the output folder	

Maya Software Renderer

The **Maya.sx** compiled Module now includes the ability to set up customized products for different renderers accessible through the Maya command line interface. Each Product is a section in the Maya.ini file which is in the Module folder. You can add your own or customize the existing ones in a similar manner to how you create Virtual Modules.

The **Maya Software** product is what used to be the normal Maya Product in Smedge 3 version 1.0.x. It actually forces the Maya command line to use the Maya Software renderer. If you are using Maya 6, you will need to use this Product if you want to render with the Maya software renderer, because the **Maya** product uses a render product switch that is not available in Maya 6. You can also use this product with Maya 7 or later if you want to force Maya to use the Maya Software renderer, or if you want the extended command line control over the render quality that you can get with the specific render products.

The Maya Software Renderer Product includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

General Information

Default ID	`44d7ee77-ffe5-4f45-abe0-8aced1a7fae7`

Type Name and Shortcuts	`Maya Software Renderer`
	`Maya Software`
	`sw`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Project	Dir	X	X	The Maya project to use for rendering. I left blank, no project will be passed to the renderer.	
RenderDir	Dir	X	X	Override the output folder	
RenumberBy	Text	X	X	The increment value for renumbering the frame files. Blank defaults to 1	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
RenumberStart	Text	X	X	The Job can be renumbered starting at this value. If left blank, the file frame numbers will correspond to the frames rendered	
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-b** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

Maya to mental ray Exporter

The **Maya.sx** compiled Module now includes the ability to set up customized products for different renderers accessible through the Maya command line interface. Each Product is a section in the Maya.ini file which is in the Module folder. You can add your own or customize the existing ones in a similar manner to how you create Virtual Modules.

The **Maya to mental ray Exporter** Product allows you to tell Maya to convert the Maya scene to .mi files for use with the mental ray standalone renderer.

The Maya to mental ray Exporter Product includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

General Information

Default ID	`17e64628-b48c-4464-86d6-3f3389758db9`

Type Name and Shortcuts	`Maya to mental ray Exporter`
	`MentalRay Exporter`
	`exporter`
	`mi`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Project	Dir	X	X	The Maya project to use for rendering. I left blank, no project will be passed to the renderer.	
RenderDir	Dir	X	X	Override the output folder	
RenumberBy	Text	X	X	The increment value for renumbering the frame files. Blank defaults to 1	
RenumberStart	Text	X	X	The Job can be renumbered starting at this value. If left blank, the file frame numbers will correspond to the frames rendered	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-b** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

Maya Vector Renderer

The **Maya.sx** compiled Module now includes the ability to set up customized products for different renderers accessible through the Maya command line interface. Each Product is a section in the Maya.ini file which is in the Module folder. You can add your own or customize the existing ones in a similar manner to how you create Virtual Modules.

The **Maya Vector Renderer** Product allows you to tell Maya to render using the Maya Vector renderer. Using this product will override any Render Layer renderer settings in your Maya 7 file.

The Maya Vector Renderer Product includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

General Information

Default ID	`fb5da2e3-22bf-41f9-adc4-e00fb6fbeed2`

Type Name and Shortcuts	`Maya Vector Renderer`
	`Maya Vector`
	`vr`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Project	Dir	X	X	The Maya project to use for rendering. I left blank, no project will be passed to the renderer.	
RenderDir	Dir	X	X	Override the output folder	
RenumberBy	Text	X	X	The increment value for renumbering the frame files. Blank defaults to 1	
RenumberStart	Text	X	X	The Job can be renumbered starting at this value. If left blank, the file frame numbers will correspond to the frames rendered	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-b** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

MayaMan

This Module is designed to control the MayaMan product. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`794874ce-eabc-485a-85a8-097b66e8b009`
Type Name and Shortcuts	`MayaMan` `mm`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Project	Dir	X	X	The Maya project to use for rendering. I left blank, no project will be passed to the renderer.	

mental ray for Maya

The **Maya.sx** compiled Module now includes the ability to set up customized products for different renderers accessible through the Maya command line interface. Each Product is a section in the Maya.ini file which is in the Module folder. You can add your own or customize the existing ones in a similar manner to how you create Virtual Modules.

The **mental ray for Maya** Product is designed to control the mental ray for Maya command line renderer included in versions Maya 6.0 and later. This product is included in the Compiled Module **Maya.sx**. Using this product will override any Render Layer renderer settings in your Maya file.

The mental ray for Maya Product includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

General Information

Default ID	`43a9184b-2b5e-4abc-958c-2aa244a36019`

Type Name and Shortcuts	`mental ray for Maya`
	`MentalRay for Maya`
	`MR4M`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Project	Dir	X	X	The Maya project to use for rendering. I left blank, no project will be passed to the renderer.	
RenderDir	Dir	X	X	Override the output folder	
RenumberBy	Text	X	X	The increment value for renumbering the frame files. Blank defaults to 1	
RenumberStart	Text	X	X	The Job can be renumbered starting at this value. If left blank, the file frame numbers will correspond to the frames rendered	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-b** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

MentalRay Standalone

The **MentalRay Standalone** Product allows you control the mental ray standalone renderer. This is a Compiled Module. It includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

This Module supports the creation of Dynamic Products with a simple INI file format. See the Dynamic Products chapter for more information.

General Information

Default ID	`bfe924c1-6d48-422c-bf71-a1305ef75437`

Type Name and Shortcuts	`MentalRay Standalone`
	`MR`
	`Ray`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
ActualScene	Text	X		Returns the full path to the actual scene file or files that will be handled in this work unit. For single .mi file scenes, it's the same as Scene, but for multiple .mi files, it will return a space separated list of files with the actual frames substituted in.	
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
SceneNameFormat	Text	X	X	For sequences of .mi files, this is the formatting string that is used to determine the scene file names. Uses standard printf formatting. Leave blank if you have the entire scene in a single .mi file.	

Nuke

This Module is designed to control Nuke via the commandline renderer. This is a Compiled Module. It includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

This Module supports the creation of Dynamic Products with a simple INI file format. See the Dynamic Products chapter for more information.

Nuke Script Path Translation

The Nuke Module can translate the paths inside of the Nuke script before rendering, using the Smedge Path translation system. This makes it easier to use Nuke across multiple platforms, because you don't have to find other ways to work around the path differences in your Read and Write nodes in Nuke. (A common way to work around this previously is to use environment variables and set the root of your path inside nuke to a script to access the environment, like [getenv PROJECTS]/rest/of/path).

When enabled (on by default), the Nuke module will copy the Nuke script file to the Job local temp folder (see the CopyLocally parameter command), and will then run through the script and translate any paths it comes across using the current Smedge path translations. It will only do this once per Engine, if needed. If there are no path translations, it will still copy the file locally, but it won't do the extra translation step. If you don't need this system, you can gain a bit of performance by disabling it, which will cause Smedge not to copy the file locally, and to start the render using the original source script directly.

This option can be set as an Engine Product option (Engine > Configure Product Options...) and can be overridden for any given Job with the parameter in the “Advanced Info” tab of the Submit Job Window.

General Information

Default ID	`dfd3ea42-d630-4326-a33b-9b9125f9b7ea`
Type Name and Shortcuts	`Nuke`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
argv	Text	X	X	Anything here can be used by [argv n] expressions to provide changing arguments to the script. Each must start with a non-digit to avoid confusion with the frame ranges. See the Nuke command line documentation for more information.	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
TranslateScriptPaths	Bool-Override	X	X	Enables the automatic Nuke script path translation system. When on, the Nuke script is copied to the job local folder and any paths inside the script are translated using the Smedge Platform path translation system	Engine Default (yes)
WriteNode	Text	X	X	An optional write node to render. If you do not supply a value for this parameter, all write nodes in the Nuke scene file will be rendered.	

RenderMan for Maya

The **Maya.sx** compiled Module now includes the ability to set up customized products for different renderers accessible through the Maya command line interface. Each Product is a section in the Maya.ini file which is in the Module folder. You can add your own or customize the existing ones in a similar manner to how you create Virtual Modules.

The **RenderMan for Maya** Product is designed to control the RenderMan for Maya plug-in for Maya. This product is included in the Compiled Module **Maya.sx**. This will override any Render Layer renderer settings in your Maya 7 file.

The RenderMan for Maya Product includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

General Information

Default ID	`6e5b4220-63ad-4a1b-b6a6-a7ccaa4ecf1a`

Type Name and Shortcuts	`RenderMan for Maya`
	`rm4m`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Project	Dir	X	X	The Maya project to use for rendering. I left blank, no project will be passed to the renderer.	
RenderDir	Dir	X	X	Override the output folder	
RenumberBy	Text	X	X	The increment value for renumbering the frame files. Blank defaults to 1	
RenumberStart	Text	X	X	The Job can be renumbered starting at this value. If left blank, the file frame numbers will correspond to the frames rendered	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-b** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

Rendition

The **Rendition** Product is designed to control the Rendition commandline renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`6cecdad0-061a-43e7-b36e-5a5f20431dcb`
Type Name and Shortcuts	`Rendition`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
FileName	File	X	X	The filename of the rendered images	

Shake

This Module is designed to control the Shake commandline renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`4c94c165-9462-498a-be5f-fb69fa8673b6`
Type Name and Shortcuts	`Shake`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	

Turtle

This Module is designed to control the Turtle for Maya commandline renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`d472685f-d229-4d9b-9e75-270d7851ea5c`

Type Name and Shortcuts	`Turtle for Maya`
	`turtle`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-frameStep** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

Viz

This Module is designed to control the Viz commandline renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`d57cf46d-3df8-4239-a163-79f2e27951bf`
Type Name and Shortcuts	`Viz`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
CommandFile	File	X	X	You can specify a file with your parameters instead of adding them manually using this option. If you use standard Smedge $(Name) syntax, you can have other parameters from the Job substituted into this filename when the commandline is actually generated	
Extra	Parameters	X	X	Extra parameters you want to send on the commandline. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-nthFrame** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

VRay for Maya

The **Maya.sx** compiled Module now includes the ability to set up customized products for different renderers accessible through the Maya command line interface. Each Product is a section in the Maya.ini file which is in the Module folder. You can add your own or customize the existing ones in a similar manner to how you create Virtual Modules.

The **VRay for Maya** Product is designed to control the VRay for Maya plug-in for Maya. This product is included in the Compiled Module **Maya.sx**. This will override any Render Layer renderer settings in your Maya 7 file.

The RenderMan for Maya Product includes all of the parameters from Job, ProcessJob, SequenceDistributor, and RenderJob.

General Information

Default ID	`31beca7c-49ea-4c14-93e9-4d0d7d225175`

Type Name and Shortcuts	`VRay for Maya`
	`vr4m`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Project	Dir	X	X	The Maya project to use for rendering. I left blank, no project will be passed to the renderer.	
RenderDir	Dir	X	X	Override the output folder	
RenumberBy	Text	X	X	The increment value for renumbering the frame files. Blank defaults to 1	
RenumberStart	Text	X	X	The Job can be renumbered starting at this value. If left blank, the file frame numbers will correspond to the frames rendered	

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
SequenceBy	Text	X	X	Allows you to change the number of frames rendered in a sequence, using the **-b** flag. Make sure that your PacketSize is set to a whole number multiple of this value. If blank, defaults to 1	

Vue

This Module is designed to control the Vue renderbull command line renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`b88c0489-128b-4e4f-9d38-f25133857745`

Type Name and Shortcuts	`VUE Renderbull`
	`Vue`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Output	Dir	X	X	The path to the rendered image files. You cannot override the filename, and there should be no slash at the end of this value.	

XSI

This Module is designed to control the XSI command line renderer. This is a Virtual Module. It includes all of the parameters from ProcessSequence.

General Information

Default ID	`b3a204c8-da3c-408b-b2c0-2d417e531ae2`
Type Name and Shortcuts	`XSI` `xsibatch`

Parameters

Name	*Type*	*Get*	*Set*	*Meaning*	*Default*
Extra	Parameters	X	X	Extra parameters you want to send on the command-line. Note: In order to be sure that the parameters are passed properly, enclose this entire list of parameters in quote marks. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	
Echo	Parameters	X	X	Custom parameters that can be set to control XSI to mental ray exporting. If you have only 1 parameter, you still need to enclose it in quote marks, and include a space somewhere in the string. (See bug database for more information.)	

Common Client Command Line Options

All Smedge client applications (everything except the Master) have a shared set of common command line options available:

`-ConnectTimeout` *float*	If no connection is established after the specified number of floating point seconds, the program will automatically terminate itself. The default is 0. Any value less than or equal to 0 will disable this time out, and the program will continue to wait for a connection indefinitely.
`-LogDisplayLevel` *integer*	If the program is running in a command prompt window, it will display log messages to that window, so that you can monitor what is going on. This switch controls what level of information is displayed. Valid values are 0 – 6, where higher numbers are more detailed display. The default display level is 4.
`-LogFileLevel` *integer*	Every application saves a log file with everything it does. This switch controls what level of information is saved in that file. Valid values are 0 – 6, where higher numbers are more detailed information. The default file level is 5.
`-LogFolder` *base-folder*	Override the machine Log folder, the base folder in which this application will create its logs.
`-Master` *master[:port]*	Override the automatic Master location system with a specific machine name and optional port.
`-MasterPort` *port*	Override the Master port only. (Default port is 6870)

SmedgeMaster Reference

SmedgeMaster is the program that performs the role of the "Master" in a Smedge environment. Normally, you don't have to interact with this process. Sometimes, however, it is useful to configure its operation. For example, if you are debugging a Virtual Module, you may want to isolate yourself from your active system.

Command Line Interface

`-AllowMultiple`	Disable the single instance check, to allow multiple instances of SmedgeMaster to run on the same machine. **Warning:** be sure to also specify an alternate database location or data corruption or crashes can result.
`-Description` *text*	*Windows only.* Used with –InstallService. This sets the "Description" text in the Service database.
`-InstallService`	*Windows only.* Tells SmedgeMaster to install itself as a service with the system.
`-License` *license_code*	Override the license code that will be used by SmedgeMaster to allow Engines to work.
`-LogDisplayLevel` *integer*	If the program is running in a command prompt window, it will display log messages to that window, so that you can monitor what is going on. This switch controls what level of information is displayed. Valid values are 0 – 6, where higher numbers are more detailed display. The default display level is 4.
`-LogFileLevel` *integer*	Every application saves a log file with everything it does. This switch controls what level of information is saved in that file. Valid values are 0 – 6, where higher numbers are more detailed information. The default file level is 5.
`-LogFolder` *directory*	Override the directory in which log files are saved.
`-MasterDatabase` *directory*	Provide an alternate path for the Master to save its data.
`-MasterPort` *port*	Override the TCP port that the Master uses for communication. (Default port is 6870.)

`-MaxJobFailures` *integer*	Override the maximum allowed Job failure count
`-MaxTypeFailres` *integer*	Override the maximum allowed Product Type failure cont.
`-ModulePath` *directory* [*directory…*]	Add additional folders to scan for Modules. You can specify as many folders as you wish. If the directory path has a space anywhere in it, you should enclose the whole path in quotes.
`-Name` *text*	*Windows only.* Used with –InstallService. This sets the "Name"of the Service in the Service database.
`-Password` *text*	*Windows only.* Used with –InstallService. This sets the password for the account that the Service will run as.
`-RemoveService`	*Windows only.*Tells SmedgeMaster to remove itself from the service database.
`-Serice`	*Windows only.* Used by the Service Manager to notify Smedge on startup that it is starting as a Service. This should not be used directly, but should be part of the commandline that starts the Service.
`-SmedgeOptionsFile` *name*	Override the name of the options file.
`-StartService`	*Windows only.* Asks the Service Control Manager to start the SmedgeMaster service.
`-StopService`	*Windows only.* Asks the Service Control Manager to stop the SmedgeMaster service.
`-User` *name*	*Windows only.* Used with –InstallService. This sets the user account that the Service will run as.
`-Wait` [*seconds*]	*Windows only*. Used with –StartService and –StopService. This requests the process to wait for the Service process to report itself as fully started or shut down before returning. If you leave off this flag, the process will return immediately after requesting the Service to close, though the operation you requested may not yet have completed. You can optionally supply a maximum number of seconds to wait. If this timeout expires before the Service has reported its status, the service may not have finished the requested operation. If you do not supply a timeout, the default timeout period is 30 seconds.

SmedgeMaster.ini options

These are all of the possible configuration values that SmedgeMaster will read from its options files. SmedgeMaster will first try to load these options from a copy of SmedgeMaster.ini in the Program folder. It will then load additional settings from a copy of SmedgeMaster.ini in the “machine” folder (on Windows this is usually C:\Documents and Settings\All Users\Application Data\Uberware\SmedgeMaster, on Linux it is usally /etc/smedge3/SmedgeMaster), overriding any settings that may have been previously loaded. The Master will then save any settings that are configured dynamically while the program is running in the machine folder.

Many Master settings can be configured directly with the Shells. More advanced settings must be manually added to the file. You should only mess with the more advanced settings under the advisement of Uberware technical support, or if you really know what you are doing. Changing the values inappropriately can cause the Master to fail to operate normally.

Note that any changes you manually to the SmedgeMaster.ini file in the Machine folder while the SmedgeMaster process is running will be lost. In order to make changes while the process is running, you must use a Shell program that can interact with the Master, such as SmedgeGui or ConfigureMaster. ConfigureMaster includes an option to copy all settings from an INI file, which would allow you to set any setting listed here while the program is running. Changing some options will require you to restart the SmedgeMaster process to have it use the new value.

```
    [ Options ]

; This stores an encrypted form of the Administrator password.
Administrate = value

; Limits the number of repeated exceptions the main thread can encounter before aborting. Default is 25
ExceptionLimit = integer

; The license code is stored here
License = license code

; Allow you to change the folder where the Master will save its data on disk
MasterDatabase = directory

; Add additional directories to look for Modules
ModulePaths = directory[;directory...]

; Change the default message displayed in the SmedgeGui about box
WelcomeMessage = text to display


    [ Communication ]

; How often to look for lost clients. Default is every 5 seconds.
CheckForLostClients = float

; Timeout period after sending messages to receive a valid response from the client. Default is 150 seconds
FinishTimeout = float

; If a client has not reported in after this amount of time since their last report, the client is considered lost
; If the client was an Engine, any work it was executing will be requeued. Default is 90 seconds
LostClientTimeout = float

; TCP and UDP ports that the Master will use to listen for clients. Default is port 6870
MasterPort = port

; Maximum number of messages to send at one time down an open TCP socket. Default is 75.
; If you set this value too high, clients may timeout if it takes longer to send the messages than the client timeout period
MaxSendAtOnce = integer

; A guide to the maximum size of the thread pools that are used for communication. Default is 10.
; The actual maximum number of threads will vary based on how many CPUs are installed on the machine.
; This setting is only read at Master startup time
MaxThreadCount = integer

; Set the polling period during communication. Default is .25 seconds
; Larger numbers use less CPU. Smaller numbers are more responsive
MessengerGranularity = float

; When a client operation cannot complete, the Master will delay trying again for this period of time
; to allow the client to finish whatever is blocking it. The default is .25 seconds
RequeueDelay = float

; Timeout period for waiting for additional messages to send on an open TCP socket. Default is .5 seconds
SendAnotherTimeout = float

; Timeout period during shutdown to wait for final communication notifications to be sent. Default is 10 seconds
ShutdownTimeout = float

; Timeout period after which inactive messenger threads will terminate to release system resources. Default is 60 seconds
ThreadIdleTimeout = float

; The scaling factor between the number of clients making concurrent contact and the number of threads to handle those clients.
; Default is 1.4
ThreadPoolScale = float

; Timeout for an incoming message to arrive on an open TCP socket. Default is 30.0 seconds
WaitMessageTimeout = float


    [ Distribution ]

; Allows work to be distributed with fewer than the requested number of CPUs
AllowFewerCPUs = boolean

; Tells the Master to automatically delete finished work as soon as it finishes
AutoDelete = boolean

; Delays distribution of work until this many seconds after the last license is checked out. Default is 5.0 seconds.
; Allows time for Engines to connect and report any work that may have completed while disconnected from the Master
; before the Master starts redistributing that work, thinking it lost.
DelayDistribution = float

; Tells the Master to automatically delete work after the specified number of hours. Ignored if AutoDelete is true.
; Default is 0, which means that the Jobs are never automatically deleted.
DeleteAfter = integer

; Amount of time to sleep between successive iterations of the distribution loop. Default is .5 seconds.
; Larger numbers use less CPU. Smaller numbers improve distribution performance.
DistributeFrequency = float

; Distribution mode. ‘true’ is First-In, First-Out; ‘false’ is Round-Robin. Default is ‘true’
FIFO = boolean
```

```
; The maximum number of work units to distribute during the global stagger start interval, regardless of the Product
; Default is 0, which means there is no limit, and no global stagger start.
GlobalStaggerCount = integer

; The global stagger start period, in seconds. Once this timeout has expired, more work can be started, up to the limit
; set in the GlobalStaggerCount above. Default is 0.0, which means there is no limit, and no global stagger start.
GlobalStaggerDelay = float

; Tells the Master to dump the dispatch loop details into the Dispatch.log file. This value will be reset
; to false at the end of a dispatch loop in which it has been set to true.
LogDispatch = boolean

; Timeout period after work has been marked as “Lost” before the work is requeued. Default is 30 seconds.
; If the Engine reconnects within this timout period, the “lost” work is found again, and is not requeued.
LostWorkTimeout = integer

; Maximum number of times an Engine is allowed to fail on a single Job. Default is 5
MaxJobFailures = integer

; Maximum number of Jobs of the same Product an Engine is allowed to fail before that Product is disabled. Default is 5.
MaxTypeFailures = integer

; Minimum elapsed work time to count as part of the average work time for a Job. Default is 5.0 seconds.
MinimumTimeForAverage = float

; Allows work to be distributed to an Engine with the requested number of CPUs, even if the Engine does not have
; that many CPUs actually available for work
OverloadEngine = boolean

; If true, no work will be distributed from Jobs with a “Priority” of 0.
; If false, priority 0 is just the lowest possible priority
PriorityZeroIsPaused = boolean

; If true, the Master will always reset the Job Creation time to the current local system time when the Job is created
; If false, the Master will keep the Job creation time that the Shell submitting the Job has set the as the Creation time
ResetCreationTime = boolean

; If true, all Engines are part of the “Whole System” pool by default. If false, there is no “Whole System” pool.
UseWholeSystem = boolean



    [ Limit ]

; Entries in this section set the limit for the total number of workers from a given type that are allowed
; to be outstanding at one time. The Syntax is <type> = <limit>. The default for a type if it is not provided
; is -1, or no limit.

typeid-string = integer


    [ Restrictions ]

; Entries in this section establish the restrictions on SmedgeGui’s functionality.
; The syntax is <restriction> = <Boolean>. If the Boolean value is true, then the restriction is in place.
; If the Boolean value is false, or if the restriction name does not appear in this section at all,
; the functionality is available to all SmedgeGui users

restriction-name = boolean
```

SmedgeEngine Reference

SmedgeEngine is the program that performs the work in a Smedge environment. Normally, you don't have to interact with this process. Sometimes, however, it is useful to configure its operation. Here is a list of the commandline parameters that SmedgeEngine understands.

`-AllowMultiple`	Disable the single instance check, to allow multiple instances of SmedgeEngine to run on the same machine. You should only do this if you are also going to override the Master or Master Port (see Common Client Commandline Options).
`-Description` *text*	*Windows only.* Used with –InstallService. This sets the “Description” text in the Service database.
`-InstallService`	*Windows only.* Tells SmedgeEngine to install itself as a service with the system.
`-ModulePath` *directory* [*directory…*]	Add additional folders to scan for Modules. You can specify as many folders as you wish. If the directory path has a space anywhere in it, you should enclose the whole path in quotes.
`-Name` *text*	*Windows only.* Used with –InstallService. This sets the “Name”of the Service in the Service database.
`-Password` *text*	*Windows only.* Used with –InstallService. This sets the password for the account that the Service will run as.
`-RemoveService`	*Windows only.*Tells SmedgeEngine to remove itself from the service database.
`-Service`	*Windows only.* Used by the Service Manager to notify Smedge on startup that it is starting as a Service. This should not be used directly, but should be part of the commandline that starts the Service.
`-SmedgeOptionsFile` *name*	Override the name of the options file.
`-StartService`	*Windows only.* Asks the Service Control Manager to start the SmedgeEngine service.
`-StopService`	*Windows only.* Asks the Service Control Manager to stop the SmedgeEngine service.

`-User` *name*	*Windows only.* Used with –InstallService. This sets the user account that the Service will run as.
`-Wait` [*seconds*]	*Windows only*. Used with –StartService and –StopService. This requests the process to wait for the Service process to report itself as fully started or shut down before returning. If you leave off this flag, the process will return immediately after requesting the Service to close, and the operation you requested may not yet have completed. You can optionally supply a maximum number of seconds to wait. If this timeout expires before the Service has reported its status, the service may not have finished the requested operation. If you do not supply a timeout, the default timeout period is 30 seconds.

Frequently Asked Questions

To better serve your needs, a large selection of searchable Frequently Asked Questions is now available on the Uberware.net web page:

http://www.uberware.net/faq.php

How do I use a Mapped Network Drive with SmedgeEngine running as a Service?

Most of the time, Smedge will handle drive remapping for you without any user interaction. If you have missing file errors that seem to be related to missing drive mapping, you can use the Mapped Drive Manager in SmedgeGui to configure multiple mapped drives for all Products on all Engines at one time. See the Mapped Drive Manager reference in the User Manual for more information.

Why isn't my Job being distributed?

When this happens, the first thing to check is if the Job or Engines have had too many errors. Once a Job or Engine hits the failure limit set in the Master Options, no more work will be distributed from that Job or to that Engine. The idea is to keep a bad Job or Engine from wasting work time.

If resetting the Job or Engine failure counts does not work, try creating a Dispatch Log. This is a log from the Master's distribution algorithm that will display the status of work distribution. From this file you should be able to debug the problem. To create a dispatch log from SmedgeGui, select System > System Commands > Generate Dispatch Log. The next time through its loop, the Master will create a file called Dispatch.log in its log folder. You must be in Administrator Mode to access this command.

Does Smedge work with disk image software?

Yes. SmedgeEngine will always determine the hardware information (Unique ID, Name, and Maximum CPUs) from the OS when it starts. Any attempt to modify this data will result in the Engine correcting itself back to the correct information. This means you do not have to worry about using disk imaging software to distribute an Engine configuration.

Additionally, SmedgeGui has the ability to configure all of the Engine settings in a single dialog box, and to load and save all Engine settings into an INI file. It also includes a customizable default system. All of this makes it easier than ever to set up new Engines.

Does Windows Firewall disrupt Smedge?

Yes. Windows Firewall blocks programs from communicating on the network. The installer tries to create the necessary exceptions to allow Smedge communication through Windows Firewall. If you have problems with connections, be sure that the firewall exceptions are enabled, or try disabling the firewall altogether to see if that is the cause of the problem.

The command to request a demo license is not there!

In order to install a license on Smedge, the SmedgeMaster needs to be running. You can then use any machine to launch the SmedgeGui and request the license. The commands to request the license will only be available if the Gui is able to connect to the Master, which must be running. The license information is all maintained and controlled by the Master, so it doesn't matter which machine you run the GUI on, because it will just get the Master's information when it connects.

However, obviously, making a connection to the runnign app is critical. So, first, make sure that the SmedgeMaster program is actually running somewhere. If you installed the Master as a Windows Service, the installer should have started the service for you. To check, you can use the Services Control Panel, and look for the Service named "SmedgeMaster". Make sure that it is started. You can also check that it is started by looking for a SmedgeMaster.exe program running in the Task Manager's list of Processes.

If the Master is installed as a service, but is not running, you can use the Services Control Panel to start it. You can also use the shortcuts that the installer created to start the service. You won't see any indication on the screen, but you should see the SmedgeMaster.exe program file appear in the Task Manager's list of Processes.

If you did not install SmedgeMaster as a Service, then you need to make sure that it is actually started and running. The Installer created a shortcut to start the Master in the Startup folder, but it does not actually start it for you. So you can use the shortcuts created by the installer to start SmedgeMaster.exe. When you do so, you should see a command prompt window with the name SmedgeMaster and a bunch of messages. You should also see SmedgeMaster.exe in the Task Manager's list of Processes.

When you start SmedgeGui, it will try to detect the Master automatically and connect to it. You can tell if the SmedgeGui is connected to the Master because you will see 6 menus instead of 3, and you will see the word "Connected" in the Status Bar. If the Master is running, but the Gui won't connect, then you have a problem connecting.

One thing to try is to use a Connection.ini file to tell the Gui where to find the Master without it having to find it automatically. The automatic detection system only works within a subnet, and uses the UDP networking transport, which is not as reliable as the TCP transport used for normal connection operations. So, using the Connection.ini file can speed up your connection time and make it more reliable. You can get more information about the Connection.ini file in the Installation Guide.

Once the GUI is connected to the Master, you should be able to request the license as described on the web site.

If you cannot get the GUI to connect to the Master, please send us the History.log files from the Master and from the GUI that will not connect to it.

My Job is immediately finished when I submit it!

For many Products, Smedge cannot determine the frame range automatically from the scene. The frame range is necessary to Smedge in order to properly manage the distribution of an animation sequence. So, you need to supply a valid range in the **Range** parameter of the Job.

Ranges can be any sequence of integers, using the **–** or **,** characters as ‘through’ and ‘and’. For example, ‘1-5’ would be the same as ‘1,2,3,4,5’ or ‘1-2,3,4-5’.

You can also check if this is your problem by looking at the Master’s History.log file. When you submit the Job, you should see an entry in the log file that looks like this:

```
[  Warning ] <Main Thread> Job 'name' has no work available, but is not marked as finished: marking as finished now
```

How do I set Environment Variables in Smedge?

If you find that your Job works normally when you execute it from a Command Prompt window, but fails in Smedge, possibly reporting missing DLLs or other missing components, this may be caused by missing or incorrect environment variables. Some Products use complicated environment setting scripts in order to work at all.

There are several ways to get environment variables set. Which way you should use will depend on how often you change the environment variables and how complicated the system to set the environment variables needs to be.

You can set the environment variables for all users. If the variables don't change too often and do not need programmatic control to set correctly, this system is the easiest.

1. Go to Control Panels->System, on the "Advanced" tab, and press the "Environment Variables" button

2. Add the appropriate environment varialbes in the "All users" section.

3. Restart the Smedge component applications running on your system.

You can also call batch files to set environment variables. This is useful when your environment variables change regularly, because you can centralize the script and only have to change one file for the changes to propogate to every machine. It is also useful for products that have complicated programmatic environment scripts that need to be used.

You can call a batch file to start the SmedgeEngine program itself, which will globally set the environment for all child processes that Smedge starts. This is useful if you change your environment regularly, or need a sophisticated script to set it properly, but can use the same environment for all the Products you use.

1. Create a text file called something like StartEngine.bat

2. You can type the environment SET commands into this file directly, or call another batch file that sets the environment variables using the CALL command. See the operating system documentation for more information on setting environment variables from a script or batch file.

3. Add a line to start the Smedge component application with all of the passed parameters at the end of the batch file. You can use syntax like this:

   ```
   SmedgeEngine %*
   ```

4. Set the shortcut or service to start SmedgeEngine (or whichever component you are setting the environment for) to use the batch file instead of starting the executable directly.

The third option is to have a batch script called to start the render executable. This way you can actually customize the environment for each Product.

1. Create a text file called something like StartProduct.bat
2. You can type the environment SET commands into this file directly, or call another batch file that sets the environment variables using the CALL command. See the operating system documentation for more information on setting environment variables from a script or batch file.
3. Add a line to start the executable for the Product with all of the passed parameters at the end of the batch file. You can use syntax like this:

   ```
   Render %*
   ```

4. Use the Configure Modules command in the Engine menu, select the Product in question, and set the **Path to the Executable** to start this batch file instead of starting the render executable directly.

Tips and Tricks

Using Engine *to control the local machine*

The new commandline shell Engine, included in Smedge 3 version 1.5.0, allows you to control any Engine with simple commands via a command prompt window, or a script. You can use this in your own network to integrate more extensive control of Smedge.

Engine allows you to specify one or more Engines, either by name or by their ID, to have the command operate on those Engines. However, it also allows you to leave off the Engine to control, in which case it tries to control an Engine running on the local machine.

This means you can create two handy shortcuts to allow users to enable or disable the Engine running on their local machine. Create a shortcut and make the commandline:

```
Engine Enable
```

(You may need to put the full path to Engine.exe if it is not in your PATH environment variable). Whenever a user launches this shortcut, it will try to enable the Engine running on their local machine. Similarly you can make another shortcut with the commandline:

```
Engine Disable
```

This will allow them to disable their machine. You can optionally specify whether the disable is immediate or delayed (immediate will attempt to terminate any currently going work, and delayed will allow it to finish). Delayed is the default if you don't specify anything, but to specify immediate make your commandline:

```
Engine Disable true
```

On Windows, if you want to use Engine Enable as part of a Log-Off script, you may want to use the Engine service mode. This mode enables the Engine shell to send the requests asynchronously, which means that when a user logs off, she doesn't have to wait for the Engine shell to connect and successfully send the request. Instead, the request is queued to be sent by a service, and the log off process can proceed more quickly. To enable this service, use the **Enable fast Log-Off script Engine control** option when you install Smedge.

Check out the Engine help by typing **`Engine`** with no parameters into the commandline.

Maintain complex Job hierarchies in a Job file

Job Files (**.sj** files) can contain any number of Jobs. When the Job File is loaded, any dependencies between jobs within the file will be correctly maintained when the file is loaded, every time it is loaded, even though the jobs will have new IDs each time.

This is useful for creating a complex render that includes preparation or post-processing steps. For example, you could have a 3-D render, then run through a composite, then make a QuickTime movie from the composite frames, making sure that each job waits for the previous to be complete before beginning.

Because the IDs are changed each time, you can repeatedly submit the same file over and over and a new set of jobs will be created each time. This is another way you can "batch submit" a bunch of Jobs at once, even if you don't need to maintain dependencies between them.