Extensions Reference

Fabric Engine Version 1.2.0-beta

Table of Contents

Chapter 1. Introduction

Extensions are additions to Fabric Engine's Kernel Language called KL and provide additional functions and types. Each extension is packaged for a particular use-case, for example the Alembic extension deals with Alembic files. Extensions can be implemented using Fabric Engine's EDK (Extension Developer Kit).

Chapter 2. Alembic Extension Guide

This extension adds a new type to KL, called the *AlembicHandle*. It is used to load Alembic.IO files. The AlembicHandle represents a file handle to an Alembic file, and can be used to retrieve data.

Table 2.1. AlembicHandle

Members	
Data pointer	The private data of the handle.
Vec2 timeRange	The min and max time of the Alembic file
Methods	
loadResource(io FabricResource resource)	Loads the content of a FabricResource onto the AlembicHandle if it doesn't contain any content yet
loadFileHandle(io String fileHandle)	Loads the Alembic file from a given FileHandle
getIdentifiers(io String identifiers[])	Loads all of the identifiers of the Alembic file. This can be used to determine how many and what kind of objects are stored in the Alembic file.
parseXform(io String identifier, io Scalar time, io Xfo transform)	Parses a Xform Alembic node at a given time and returns the Fabric Engine Xfo.
parseCamera(io String identifier, io Scalar time, io Scalar near, io Scalar far, io Scalar fovY)	Parses a Camera Alembic node at a given time and returns near and far clipping, as well as vertical field of view.
parsePolyMeshCount(io String identifier, io Size count)	Parses a PolyMesh Alembic node and returns the number of vertices of that mesh.
parsePolyMeshUniforms(io String identifier, io Integer indices[])	Parses a PolyMesh Alembic node and returns the face indices as a triangles list.
parsePolyMeshAttributes(io String identifier, io Scalar time, io Vec3 vertices<>, io Vec3 normals<>, io Boolean uvsLoaded, io Vec2 uvs<>)	Parses a PolyMesh Alembic node at a given time and returns the mesh's vertices, normals, and optionally, uvs. All arrays are per vertex data.
parsePointsCount(io String identifier, io Scalar time, io Size count)	Parses a Points Alembic node at a given time and returns its point count
parsePointsAttributes(io String identifier, io Scalar time, io Vec3 positions<>, io Quat orientations<>, io Scalar sizes<>, io Vec3 scales<>, io Color colors<>)	Parses a Points Alembic node at a given time and returns its positions, orientations, sizes, scales and colors. All arrays are per vertex data.
parseCurvesCount(io String identifier, io Size count)	Parses a Curves Alembic node at a given time and returns its vertex count
parseCurvesUniforms(io String identifier, io Integer indices[])	Parses a Curves Alembic node and returns its line indices (as a from-to index list)
parseCurvesAttributes(io String identifier, io Scalar time, io Vec3 vertices<>, io Scalar sizes<>, io Boolean uvsLoaded, io Vec2 uvs<>, io Color colors<>)	Parses a Curves Alembic node at a given time and returns its vertices, sizes, uvs and colors. All arrays are per vertex data.

Chapter 3. Bullet Extension Guide

The Bullet Physics extension adds support for physics simulation in KL. It implements all of the core Bullet types, and allows perform raycasting into the bullet scene. For additional information please refer to the Bullet Physics Wiki (http://bulletphysics.org/mediawiki-1.5.8/index.php/Main_Page)

Table 3.1. BulletWorld

Members	
Data localData	The private data of the Bullet type.
Vec3 gravity	The gravity applied during the simulation.
Size step	The current simulation step of the simulation.
Size substeps	The number of substeps to perform per simulation step.
Boolean hit	Indicates, after a raycast has been performed, if there is any hit object.
Methods	
create(io Boolean success)	Creates the simulation world and returns if successful.
step(io Scalar timeStep)	Steps through the simulation for a provided timeStep in seconds.
reset()	Resets the simulation world by moving all rigid and soft bodies to their initial state and setting the simulation's step to 0.
raycast(io Vec3 from, io Vec3 to, io Boolean filterPassiveObjects, io BulletContact contacts[])	Performs a raycast into the simulation world and returns the hit contacts.
applyForce(io BulletForce force)	Applies a force to the simulation at the current simulation step.
getGravity(io Vec3 gravity)	Returns the current gravity of the simulation.
setGravity(io Vec3 gravity)	Sets the gravity of the simulation.
addRigidBody(io BulletRigidBody body)	Adds a rigid body to the simulation.
removeRigidBody(io BulletRigidBody body)	Removes a rigid body from the simulation.
addSoftBody(io BulletSoftBody body)	Adds a soft body to the simulation.
removeSoftBody(io BulletSoftBody body)	Removes a soft body from the simulation.
addConstraint(io BulletConstraint constraint)	Adds a constraint to the simulation.
removeConstraint(io BulletConstraint constraint)	Removes a constraint from the simulation.

Table 3.2. BulletShape

Members	
Data localData	The private data of the Bullet type.
Integer type	The type of the shape. The Bullet extension also provides constants for this type. The valid values are: BULLET_BOX_SHAPE, BULLET_CONVEX_HULL_SHAPE, BULLET_SPHERE_SHAPE, BULLET_CAPSULE_SHAPE, BULLET_CONE_SHAPE, BULLET_CYLINDER_SHAPE, BULLET_TRIANGLEMESH_SHAPE,

	BULLET_GIMPACT_SHAPE, BULLET_PLANE_SHAPE and BULLET_COMPOUND_SHAPE.
String name	The name of the shape. Names have to be unique.
Scalar parameters[]	The parameter for the shape creation. The number of parameters varies for each shape. To determine which parameters should be used create a shape without parameters and read the thrown exception.
Vec3 vertices[]	For convex hull, gimpact or triangle mesh this stores the vertices of the shape. For all other shape types it isn't used.
Integer indices[]	For gimpact or triangle mesh shapes this stores the triangle indices of the shape. For all other shape types it isn't used.
Methods	
create(io Boolean success)	Creates the shape and returns if successful.

Table 3.3. BulletRigidBody

Members	
Data localData	The private data of the Bullet type.
String name	The name of the rigid body. Names have to be unique.
Xfo transform	The initial transform.
Scalar mass	The mass in kilograms. A mass of 0.0 indicates a passive, non-simulated rigid body.
Scalar friction	The combined dynamic and static friciton (0.0 to 1.0).
Scalar restitution	The restitution factor, from 0.0 to 1.0
Methods	
create(io BulletShape shape, io Boolean success)	Creates the rigid body based on its member data and the provided shape, returns if successful.
setMass(in Scalar mass)	Sets the mass
Xfo getTransform()	Returns the current simulated transform.
setTransform(in Xfo transform)	Sets the transform. This only works for passive rigid bodies (mass of 0.0)
Vec3 getLinearVelocity()	Returns the current linear velocity
setLinearVelocity(in Vec3 velocity)	Sets the linear velocity
Vec3 getAngularVelocity()	Returns the angular velocity (as euler angles in radians)
setAngularVelocity(in Vec3 velocity)	Sets the angular velocity (from euler angles in radians)

Table 3.4. BulletSoftBody

Members	
Data localData	The private data of the Bullet type.
String name	The name of the soft body. Names have to be unique.
Xfo transform	The initial transform.
Integer clusters	The cluster count to use for this softbody. When set to zero, the cluster collision algorithm won't be used, and a per vertex spring model will be used.

Integer constraints	The recursion depth level for bending constraints. Set to 1 there will be a spring between each neighbor vertex, set to 2 introduces springs between neighbors of neighbors etc.
Scalar mass	The mass of the softbody in kilograms.
Scalar stiffness	The linear stiffness factor for the springs (0.0 to 1.0)
Scalar friction	The dynamic friction of each softbody vertex (0.0 to 1.0)
Scalar conservation	The volume conservation of the softbody (0.0 to 1.0). This is a very sensitive parameter and use of values higher than 0.1 is not recommended.
Scalar pressure	The internal pressure of the soft body in nm.
Scalar recover	The amount of shape recovery (0.0 to 1.0). If set to 0.5, for example, the shape is blended back to its original shape by 50%.

Methods

create(io Boolean success, io BulletWorld world, io Vec3 positions<>, io Vec3 normals<>, io Integer indices[])	Creates the softbody in a provided simulation world, using the provided positions, normals and indices arrays (triangles). Returns if successful.
getPosition(in Size index, io Vec3 position, io Vec3 normal)	Returns the current position and normal for a given vertex index.

Table 3.5. BulletConstraint

Members

Data localData	The private data of the Bullet type.
Data bodyLocalDataA	The pointer to the private data of the first attached rigid body.
Data bodyLocalDataB	The pointer to the private data of the second attached rigid body.
Integer type	The type of the constraint. Valid values are 3 (point2point), 4 (hinge) and 7 (slider).
String name	The name of the constraint. Names have to be unique.
Xfo pivotA	The pivot transform in local space for the first attached rigid body
Xfo pivotB	The pivot transform in local space for the second attached rigid body
String nameA	The name of the first attached rigid body.
String nameB	The name of the second attached rigid body.
Integer indexA	The index of the first attached rigid body.
Integer indexB	The index of the second attached rigid body.
Scalar parameters[]	The parameters for the constraint creation. This member is obsolete and is not being used.

Methods

create(io Boolean result, io BulletRigidBody bodiesA[], io BulletRigidBody bodiesB[])	Creates the constraing based on its *indexA* and *indexB* members. The private data pointers are set by retreiving them from A and B rigid body arrays. This allows to construct a large number of constraints on lists of rigid bodies.

Table 3.6. BulletForce

Members

String name	The name of the force. Names have to be unique.
Vec3 origin	The point the force is coming from (global space).
Vec3 direction	The direction of the force (global space)
Scalar radius	The influence radius of the force
Scalar factor	The force factor in nm.
Boolean useTorque	Determines if the force should apply rotation or only linear velocity.
Boolean useFalloff	If set the force will use a linear falloff inside its radius.
Boolean enabled	If set to false the force will be ignored
Boolean autoDisable	If set to true the force's *enabled* flag will be set once it is applied. This is useful for one-shot forces, based on mouse clicks, for example.

Table 3.7. BulletAnchor

Members

Data localData	The private data of the Bullet type.
Data rigidBodyLocalData	The pointer to the private data of the attached rigid body.
Data softBodyLocalData	The pointer to the private data of the attached soft body.
String name	The name of the anchor. Names have to be unique.
Integer rigidBodyIndex	The index of the attached rigid body.
Integer softBodyNodeIndices[]	The indices of the attached vertices of the soft body mesh.
Boolean disableCollision	Determines if the rigid body and soft body attached to the anchor should intercollide.

Methods

create(io Boolean success, io BulletRigidBody rigidBodies[], io BulletSoftBody softBody)	Creates the constraing based on its *rigidBodyIndex*. The private data pointers are set by retreiving them from the rigid body array resp. the provided soft body. This allows to construct a large number of anchors on a list of rigid bodies. Returns if successful.

Table 3.8. BulletContact

Members

Scalar fraction	The fraction of the ray
Vec3 normal	The normal of the hit surface position
Scalar mass	The mass of the collision object's surface position.
Vec3 linearVelocity	The linear velocity of the collision object.
Vec3 angularVelocity	The angular velocity of the collision object (euler angles in radians).

Chapter 4. CIMG Extension Guide

The CIMG extension wraps the CIMG C++ image library (http://cimg.sourceforge.net) and provides read and write access to image within Fabric Engine. The extension doesn't provide any types, but KL functions to perform the image IO.

Table 4.1. CIMG Functions

FabricCIMGDecode(Data data, Size dataSize, io String ext, io Size imageWidth, io Size imageHeight, io RGBA imagePixels[])	Decodes an image, stored as an encoded Byte array into its width, height and pixel values. The extention has to be provided to inform CIMG what kind of image is stored in the Byte array.
FabricCIMGOpenFileHandle(String fileHandle, io String ext, io Size imageWidth, io Size imageHeight, io RGBA imagePixels[])	Opens an image from a provided FileHandle and reads the image data into the width, height and pixel values. The extention has to be provided to inform CIMG what kind of image is stored in the FileHandle.
FabricCIMGCreateFromText(String text, io Size imageWidth, io Size imageHeight, io RGBA imagePixels[])	Encodes the text provided into a new image resulting in width, height and pixel values. The image uses a fixed text size and is stored as black and white, where white is the text and black is the background.
FabricCIMGSaveToFileHandle(String fileHandle, Size imageWidth, Size imageHeight, Boolean mirrorVertically, io RGBA imagePixels[])	Saves a provided image, represented by with, height and pixels into a provided writable FileHandle. If required, the image can be flipped vertically prior to saving.

Chapter 5. FILESTREAM Extension Guide

The FILESTREAM extension is a wrapper for standard file IO functionality, plus extra features such as reading or writing to compressed formats. The *FabricFileStream* type wraps an open file handle. A *FabricFileStream* is initialized from a *FabricFileHandle String*, which can be an abstract handle or a direct file path depending on the client and its security model (see Fabric IO programming guide).

Table 5.1. FabricFileStream

Members	
Data m_data	The private data of the *FabricFileStream* type.
Methods	
open(in String handle, in String mode)	Opens the file associated to the *FabricFileHandle handle*. Mode can be "r" for read-only, "w" for write, and "a" for append. If "w" or "a", the handle must have a write permission.
close()	Closes the file; the *FabricFileStream* is invalid unless *open* is called again.
closeOnFullyRead(in Boolean close)	Enables a special mode in which *close* will be called implicitely once all the file was read.
Boolean isValid()	Returns *true* if the *FabricFileStream* was opened successfully
Boolean isWritable()	Returns *true* if the *FabricFileStream* was opened with a writable mode ("w" or "a")
Size getSize()	Returns the file size (byte count).
Size getSizeRead()	Returns the total number of bytes that was read.
Size getSeek()	Returns the current file read or write position.
setSeek(in Size seek)	Sets the read or write file position.
setSeekStart()	Sets the read or write position at the start of the file.
writeData(in Data data, in Size size)	Writes *size* bytes from the *data* buffer.
readData(in Data data, in Size size)	Reads *size* bytes to the *data* buffer.
writeDataCompressed(in Data data, in Size size)	Compresses and writes *size* bytes from the *data* buffer. Later, these bytes should be read with the *readDataCompressed* method.
readDataCompressed(in Data data, in Size size)	Uncompresses and reads *size* bytes to the *data* buffer. Originally, these bytes should have been written with the *writeDataCompressed* method.

Chapter 6. FILESYSTEM Extension Guide

The FILESYSTEM extension enables to browse and modify the local file system. Because it enables unsecure operations, this extension must be installed separately for Fabric browser plug-in clients. This extension's *FabricFolderHandle* type wraps folders and *FabricFileHandleWrapper* type wraps a *FabricFileHandle* (see Fabric IO programming guide). Note that the FILESTREAM extension initializes its FabricFileStream from a *FabricFileHandle*, which can be retrived from *FabricFileHandleWrapper*'s *getHandle* method.

Table 6.1. FabricFolderHandle

Members	
Data m_data	The private data of the FabricFolderHandle type.
Methods	
setAbsolutePath(in String path)	Sets the absolute path of the folder wrapper. No validation is done; call *exists* to know if it exists.
getAbsolutePath(io String result)	Gets the absolute path of the folder wrapper.
isValid(io Boolean result)	Returns *true* if the wrapper is associated to an existing folder.
isValid(io Boolean result)	Result is set to *true* if the absolute path was set.
exists(io Boolean result)	Result is set to *true* if the associated folder exists.
getParentFolder(io FabricFolderHandle result)	Result is set to the parent folder wrapper.
getSubFolders(io FabricFolderHandle subfolders[])	*subfolders* will contain wrappers for all sub folder.
getFiles(io FabricFileHandleWrapper files[])	*files* will contain wrappers for all sub files.
createFolder()	Creates a folder for the path that was set with *setAbsolutePath*.

Table 6.2. FabricFileHandleWrapper

Members	
String m_handle	Contains a *FabricFileHandle String*, which can be an abstract handle or a direct file path depending on the client and its security model (see Fabric IO programming guide).
Methods	
String getHandle()	Simply returns the underlying *FabricFileHandle* (*m_handle* member).
setHandle(in String handle)	Sets the wrapped *FabricFileHandle* (*m_handle* member).
setAbsolutePath(in String path)	Builds a *FabricFileHandle* associated to the absolute path and puts it in *m_handle*.
getAbsolutePath(io String path)	Returns the absolute path associated to the *m_handle FabricFileHandle*.
getParentFolder(io FabricFolderHandle result)	Returns the parent folder FabricFolderHandle wrapper.
getName(io String name)	Returns the name of the file associated with *m_handle*.
getBaseName(io String baseName)	Returns the name without extension of the file associated with *m_handle*.
getExtension(io String extension)	Returns the extension of the file associated with *m_handle*.

getExtensionLower(io String extensionLower)	Returns the lower-case extension of the file associated with *m_handle*.
isValid(io Boolean result)	*result* is set to *true* if *m_handle* is a valid *FabricFileHandle*.
exists(io Boolean result)	*result* is set to *true* if *m_handle* is associated to an existing file.
isReadOnly(io Boolean result)	*result* is set to *true* if *m_handle* has no write permission.
getSize(io Size result)	*result* is set to the size (byte count) of the file associated with *m_handle*.

Chapter 7. LIDAR Extension Guide

The LIDAR extension is a wrapper for the liblas library (http://liblas.org/). It provides a type to read the contents of a LIDAR file and use it inside Fabric Engine.

Table 7.1. LidarReader

Members	
Data pointer	The private data of the LidarReader type.
String url	The url of the parsed lidar file.
Boolean compressed	After the file is opened this will indicate if it is a compressed file or not.
Methods	
loadResource(io FabricResource resource, io String url)	Loads a lidar file stored in memory. The url value is not used in this case, and is just cosmetic.
loadFileHandle(in String handle)	Loads a lidar file stored in a readable FIleHandle.
getCount(io Size count)	Returs the number of points in the lidar file.
getPoints(io Vec3 positions<>, io Color colors<>)	Returns all of the point positions and colors inside the lidar file. If the lidar file doesn't contain any colors, they will be all black.

Chapter 8. Math Extension Guide

The Math extension provides additional Math features to KL. Currently it only contains a pseudo random number generator.

Table 8.1. Math Functions

Integer mathRandomInteger(in Size id, in Size offset)	Returns the random integer number id with a provided random offset. The offset can be understood as the seed, while the id is the index of the random number in the sequence. The range is the full integer range.
Scalar mathRandomScalar(in Size id, in Size offset)	Returns the random scalar number id with a provided random offset. The offset can be understood as the seed, while the id is the index of the random number in the sequence. The range is the full scalar range.

Chapter 9. OBJ Extension Guide

The OBJ extension provides an OBJ parser to Fabric Engine. It is implemented as a simple type storing the handle to the parser, and KL functions allowing to query the parser.

Table 9.1. OBJDataHandle

Members	
Data handle	The private data of the OBJDataHandle type.

Table 9.2. OBJ Functions

FabricOBJDecode(Data objData, Size objDataSize, Boolean splitByObjects, Boolean splitByGroups, Boolean splitByMaterials, io OBJDataHandle handle)	Parses an OBJ file stored in memory as a Data pointer of a given size. splitByObjects determines if objects should be merged or represented as separate objects, splitByGroups determines is shading groups should be split into separate objects or not, and splitByMaterial determines if per face shading should be resulting in separate objects or not.
FabricOBJOpenFileHandle(String fileHandle, Boolean splitByObjects, Boolean splitByGroups, Boolean splitByMaterials, io OBJDataHandle handle)	Parses an OBJ file stored in a readable FileHandle. splitByObjects determines if objects should be merged or represented as separate objects, splitByGroups determines is shading groups should be split into separate objects or not, and splitByMaterial determines if per face shading should be resulting in separate objects or not.
FabricOBJIsHandleValid(OBJDataHandle handle, io Boolean valid)	Checks if a provided OBJDataHandle is valid.
FabricOBJFreeParsedData(io OBJDataHandle handle)	Frees the parsed data from memory. This is useful if the parser is no longer required.
FabricOBJHadErrors(OBJDataHandle handle, io Boolean hadErrors)	Checks if the parse contained any errors.
FabricOBJGetErrors(OBJDataHandle handle, io String errors[])	Returns the errors which happened during the parse.
FabricOBJHasTextureCoords(OBJDataHandle handle, io Boolean hasTextureCoords)	Checks if the parsed OBJ file contains texture coordinates.
FabricOBJGetMaterialLibraries(OBJDataHandle handle, io String names[])	Returns the names of the material libraries contained in the parsed OBJ file.
FabricOBJGetMaterialNames(OBJDataHandle handle, io String names[])	Returns the names of the materials contained in the parsed OBJ file.
FabricOBJGetNbEntities(OBJDataHandle handle, io Size nbEntities)	Returns the number of entities in the parsed OBJ file.
FabricOBJGetEntityObjectName(OBJDataHandle handle, Integer entity, io String name)	Returns the object name of a given entity.
FabricOBJGetEntityGroupName(OBJDataHandle handle, Integer entity, io String name)	Returns the group name of a given entity.
FabricOBJGetEntityMaterialName(OBJDataHandle handle, Integer entity, io String name)	Returns the material name of a given entity.
FabricOBJGetNbEntityPoints(OBJDataHandle handle, Integer entity, io Size nbPoints)	Returns the number of points for a given entity

FabricOBJGetEntityPoints(OBJDataHandle handle, Integer entity, io Vec3 points[])	Returns the point positions of a given entity as a variable array
FabricOBJGetEntityPointsSliced(OBJDataHandle handle, Integer entity, io Vec3 points<>)	Returns the point positions of a given entity as a sliced array
FabricOBJGetEntityNormals(OBJDataHandle handle, Integer entity, io Vec3 normals[])	Returns the point normals of a given entity as a variable array
FabricOBJGetEntityNormalsSliced(OBJDataHandle handle, Integer entity, io Vec3 normals<>)	Returns the point normals of a given entity as a sliced array
FabricOBJGetEntityTextureCoords(OBJDataHandle handle, Integer, io Vec2 texCoords[])	Returns the point texture coordinates of a given entity as a variable array
FabricOBJGetEntityTextureCoordsSliced(OBJDataHandle handle, Integer entity, io Vec2 texCoords<>)	Returns the point texture coordinates of a given entity as a sliced array
FabricOBJGetNbEntityTriangles(OBJDataHandle handle, Integer entity, io Size nbTriangles)	Returns the number of triangles of a given entity
FabricOBJGetEntityTriangleIndices(OBJDataHandle handle, Integer entity, io Integer triangleIndices[])	Returns the triangle indices of a given entity as a variable array
FabricOBJGetEntityTriangleIndicesSliced(OBJDataHandle handle, Integer entity, io Integer triangleIndices<>)	Returns the triangle indices of a given entity as a sliced array
FabricOBJGetEntityTriangleMaterialIndices(OBJDataHandle handle, Integer entity, io Integer triangleIndices[])	Returns the triangle material indices of a given entity as a variable array
FabricOBJGetEntityTriangleMaterialIndicesSliced(OBJDataHandle handle, Integer entity, io Integer triangleIndices<>)	Returns the triangle material indices of a given entity as a sliced array

Chapter 10. OGL Extension Guide

The OGL extension provides all valid OpenGL functions to KL. It acts as a wrapper for Glew (http://glew.sourceforge.net/), essentially. With the OGL extension you can use any OpenGL call inside KL. For example:

```
use FabricOGL;

glClearColor(1.0, 0.0, 0.0, 1.0);
glEnable(GL_CULL_FACE);
glPatchParameteri(GL_PATCH_VERTICES, 3);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
```

Please see any of the SceneGraph KL files for references of this. All of the SceneGraph's rendering has been implemented using this extension.

Chapter 11. OPENCV Extension Guide

The OpenCV FACE extension is a wrapper for the OpenCV library (http://opencv.willowgarage.com/wiki). Currently only a very small subset of the functionality is exposed, mainly focusing on face detection.

Table 11.1. FaceLocation

Members	
Size left	Left x coordinate of the face rectangle
Size right	Right x coordinate of the face rectangle
Size top	Top y coordinate of the face rectangle
Size bottom	Bottom y coordinate of the face rectangle

Table 11.2. FaceDetector

Members	
Data pointer	The private data of the FaceDetector type.
Methods	
init(in String fileHandle)	Initiates a face detector using a haarcascade xml file. The haarcascade describes the face detection method. Please refer to the OpenCV documentation for more details.
detectRGB(io RGB pixels[], Size width, Size height, Scalar ratio, Size searchWidth, Size searchHeight, io FaceLocation faces[])	Detect faces in a RGB provided image with pixels, width and height. Ratio is used for the frame to frame move different (1.5 is the default), searchWidth and searchHeight define the minimum size of a face in pixels. Found faces are returned as an array of FaceLocation.
detectRGBA(io RGBA pixels[], Size width, Size height, Scalar ratio, Size searchWidth, Size searchHeight, io FaceLocation faces[])	Detect faces in a RGBA provided image with pixels, width and height. Ratio is used for the frame to frame move different (1.5 is the default), searchWidth and searchHeight define the minimum size of a face in pixels. Found faces are returned as an array of FaceLocation

Chapter 12. VIDEO Extension Guide

The VIDEO extension is a wrapper for the ffmeg library (http://ffmpeg.org/). It provides a type to read as well as write video. The VIDEO extension currently doesn't support audio streams.

Table 12.1. VideoHandle

Members	
Data pointer	The private data of the VideoHandle type.
Size width	The width of the video.
Size height	The height of the video.
Scalar duration	The duration of the video in seconds.
Scalar fps	The framerate of the video (frames per seconds).
Scalar time	The current time of the video.

Table 12.2. VIDEO Functions

FabricVIDEOOpenResource(Data resourceData, Size resourceDataSize, io VideoHandle handle)	Opens a video handle for reading from memory. The video is stored as a pointer with a provided size.
FabricVIDEOOpenFileHandle(String fileHandle, io VideoHandle handle)	Opens a video handle for reading from a readable FileHandle.
FabricVIDEOCreateFromFileHandle(String file, Size width, Size height, io VideoHandle handle,)	Creates a video handle for writing from a writable FileHandle, with the provided width and height.
FabricVIDEOFreeHandle(io VideoHandle handle)	Closes the video handle.
FabricVIDEOSeekTime(io VideoHandle handle, io Scalar time)	Seeks to a provided time in the video. This only works for reading video handles.
FabricVIDEOGetAllPixels(io VideoHandle handle, io RGB pixels[])	Returns all RGB pixels of the current video frame. This only works for reading video handles.
FabricVIDEOWriteAllPixels(io VideoHandle handle, io RGB pixels[])	Writes the provided RGB pixels as a new frame to the video handle. This only works with writing video handles.