JanICE documentation

version 4.0.1

Noblis

October 10, 2017

Contents

Welcome to JanICE's documentation!

Concepts

Error Handling

The API handles errors using return codes. Valid return codes are defined JaniceError. In general, it is assumed that new memory is only allocated if a function returns JANICE_SUCCESS. Therefore, **implementors are REQUIRED to deallocate any memory allocated during a function call if that function returns an error.**

Memory Allocation

The API often passes unallocated pointers to functions for the implementor to allocate appropriately. This is indicated if the type of a function input is JaniceObject**, or in the case of a utility typedef JaniceTypedef*. It is considered a best practice for unallocated pointers to be initialized to NULL before they are passed to a function, but this is not guaranteed. It is the responsibility of the users of the API to ensure that pointers do not point to valid data before they are passed to functions in which they are modified, as this would cause memory leaks.

Thread Safety

All functions are marked one of:

Thread Safe

Can be called simultaneously from multiple threads, even when the invocations use shared data.

Reentrant

Can be called simultaneously from multiple threads, but only if each invocation uses its own data.

Thread Unsafe

Can not be called simultaneously from multiple threads.

Compiling

Define JANICE_LIBRARY during compilation to export JanICE symbols.

Versioning

This API follows the semantic versioning paradigm. Each released iteration is tagged with a major.minor.patch version. A change in the major version indicates a breaking change. A change in the minor version indicates a backwards-compatible change. A change in the patch version indicates a backwards-compatible bug fix.

Errors

Overview

Every function in the JanICE *C* API returns an error code when executed. In the case of successful application JANICE_SUCCESS is returned, otherwise a code indicating the specific issue is returned. The error codes are enumerated using the JaniceError type.

Enumerations

JaniceError

The error codes defined in the JanICE *C* API

Code	Description
JANICE_SUCCESS	No error
JANICE_UNKNOWN_ERROR	Catch all error code
JANICE_OUT_OF_MEMORY	Out of memory error
JANICE_INVALID_SDK_PATH	Invalid SDK location
JANICE_BAD_SDK_CONFIG	Invalid SDK configuration
JANICE_BAD_LICENSE	Incorrect license file
JANICE_MISSING_DATA	Missing SDK data
JANICE_INVALID_GPU	The GPU is not functioning
JANICE_BAD_ARGUMENT	An argument to a JanICE function is invalid
JANICE_OPEN_ERROR	Failed to open a file
JANICE_READ_ERROR	Failed to read from a file
JANICE_WRITE_ERROR	Failed to write to a file
JANICE_PARSE_ERROR	Failed to parse a file
JANICE_INVALID_MEDIA	Failed to decode a media file
JANICE_OUT_OF_BOUNDS_ACCESS	Out of bounds access into a buffer.
JANICE_MEDIA_AT_END	A media iterator has reached the end of its data.
JANICE_INVALID_ATTRIBUTE_KEY	An invalid attribute key was provided.
JANICE_MISSING_ATTRIBUTE	A value for a valid attribute key is not available.
JANICE_DUPLICATE_ID	Template id already exists in a gallery
JANICE_MISSING_ID	Template id can't be found
JANICE_MISSING_FILE_NAME	An expected file name is not given
JANICE_INCORRECT_ROLE	Incorrect template role
JANICE_FAILURE_TO_SERIALIZE	Could not serialize a data structure
JANICE_FAILURE_TO_DESERIALIZE	Could not deserialize a data structure
JANICE_NOT_IMPLEMENTED	Optional function return
JANICE_NUM_ERRORS	Utility to iterate over all errors

Functions

janice_error_to_string

Convert a JaniceError into a string for printing.

Signature

```
JANICE_EXPORT const char* janice_error_to_string(JaniceError error);
```

Thread Safety

This function is Thread Safe.

Parameters

Name	Type	Description
error	JaniceError	An error code

Return Value

This is the only function in the API that does not return JaniceError. It returns const char* which is a null-terminated list of characters that describe the input error.

Initialization

Functions

janice_initialize

Initialize global or shared state for the implementation. This function should be called once at the start of the application, before making any other calls to the API.

Signature

```
JANICE_EXPORT JaniceError janice_initialize(const char* sdk_path,
                                            const char* temp_path,
                                            const char* algorithm,
                                            const int num_threads,
                                            const int* gpus,
                                            const int num_gpus);
```

Thread Safety

This function is Thread Unsafe.

Parameters

Name	Type	Description
sdk_path	const char*	Path to a *read-only* directory containing the JanICE compliant SDK as specified by the implementor.
temp_path	const char*	Path to an existing empty *read-write* directory for use as temporary file storage by the implementation. This path must be guaranteed until janice_finalize.
algorithm	const char*	An empty string indicating the default algorithm, or an implementation defined containing an alternative configuration.
num_threads	const int	The number of threads the implementation is allowed to use. A value of '-1' indicates that the implementation should use all available hardware.
gpus	const int*	A list of indices of GPUs available to the implementation. The length of the list is given by *num_gpus*. If the implementor does not require a GPU in their solution they can ignore this parameter.
num_gpus	const int	The length of the *gpus* array. If no GPUs are available this should be set to 0.

janice_api_version

Query the implementation for the version of the JanICE API it was designed to implement. See Versioning for more information on the versioning convention for this API.

Signature

```
JANICE_EXPORT JaniceError janice_api_version(uint32_t* major,
                                             uint32_t* minor,
                                             uint32_t* patch);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
major	uint32_t*	The supported major version of the API.
minor	uint32_t*	The supported minor version of the API.
patch	uint32_t*	The supported patch version of the API.

janice_sdk_version

Query the implementation for its SDK version.

Signature

```
JANICE_EXPORT JaniceError janice_sdk_version(uint32_t* major,
                                             uint32_t* minor,
                                             uint32_t* patch);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
major	uint32_t*	The major version of the SDK.
minor	uint32_t*	The minor version of the SDK.
patch	uint32_t*	The patch version of the SDK.

janice_finalize

Destroy any resources created by janice_initialize and finalize the application. This should be called once after all other API calls.

Signature

```
JANICE_EXPORT JaniceError janice_finalize();
```

Thread Safety

This function is Thread Unsafe.

I/O

Overview

As a computer vision API it is a requirement that images and videos are loaded into a common structure that can be processed by the rest of the API. In this case, we strive to isolate the I/O functions from the rest of the API. This serves three purposes:

1. It allows implementations to be agnostic to the method and type of image storage, compression techniques and other factors
2. It keeps implementations from having to worry about licenses, patents and other factors that can arise from distributing proprietary image formats
3. It allows implementations to be “future-proof” with regards to future developments of image or video formats

To accomplish this goal the API defines a simple interface of two structures, JaniceImageType and JaniceMediaIteratorType which correspond to a single image or frame and an entire media respectively. These interfaces allow pixel-level access for implementations and can be changed independently to work with new formats.

Structs

JaniceImageType

A structure representing a single frame or an image

Fields

Name	Type	Description
channels	uint32_t	The number of channels in the image.
rows	uint32_t	The number of rows in the image.
cols	uint32_t	The number of columns in the image.
data	uint8_t*	A contiguous, row-major array containing pixel data.
owner	bool	True if the image owns its data and should delete it, false otherwise.

JaniceImage

A pointer to a JaniceImageType.

Signature

```
typedef struct JaniceImageType* JaniceImage;
```

JaniceMediaIteratorState

A void pointer to a user-defined structure that contains state required for a JaniceMediaIteratorType.

JaniceMediaIteratorType

An interface representing a single image or video. JaniceMediaIteratorType implements an iterator interface on media to enable lazy loading via function pointers.

Fields

Name	Type	Description
next	JaniceError(JaniceMediaIteratorType*, JaniceImage*)	A function pointer that advances the iterator one frame. The next video frame or image should be stored in the JaniceImage parameter. If the next frame or image is stored in the image parameter this function should return *JANICE_SUCCESS*.
seek	JaniceError(JaniceMediaIteratorType*, uint32_t)	A function pointer that advances the iterator to a specific frame. If the iterator is a video and the seek is successful, this function should return *JANICE_SUCCESS*. If the iterator is an image, this function should return *JANICE_NOT_IMPLEMENTED*.
get	JaniceError(JaniceMediaIteratorType*, JaniceImage*, uint32_t)	A function pointer that advances the iterator to a specific frame and stores that frame in the JaniceImage parameter. If the iterator is a video and the seek and subsequent retreival are successful, this function should return *JANICE_SUCCESS*. If the iterator is an image, this function should return *JANICE_NOT_IMPLEMENTED*.
tell	JaniceError(JaniceMediaIteratorType*, uint32_t*)	A function pointer that reports the current position of the iterator. If the iterator is a video and the current position is successfully queried this function should return *JANICE_SUCCESS*. If the iterator is an image, this function should return *JANICE_NOT_IMPLEMENTED*.
reset	JaniceError(JaniceMediaIteratorType*)	A function that resets an iterator to an initial, valid state.
free_image	JaniceError(JaniceImage*)	A function pointer to free a JaniceImage object.
free	JaniceError(JaniceMediaIteratorType**)	A function pointer to free a JaniceMediaIteratorType object.

Typedefs

JaniceMediaIterator

A pointer to a JaniceMediaIteratorType object.

Signature

```
typedef struct JaniceMediaIteratorType* JaniceMediaIterator;
```

JaniceMediaIterators

A structure representing a list of JaniceMediaIterator objects.

Fields

Name	Type	Description
media	JaniceMediaIterator*	An array of media iterator objects.
length	size_t	The number of elements in *media*

Context

A context is a single object for managing the various hyperparameters parameters required by JanICE functions.

Enumerations

JaniceDetectionPolicy

A policy that controls the types of objects that should be detected by a call to janice_detect. Supported policies are:

Policy	Description
JaniceDetectAll	Detect all objects present in the media.
JaniceDetectLargest	Detect the largest object present in the media. Running detection with this policy should produce at most one detection.
JaniceDetectBest	Detect the best object present in the media. The implementor is responsible for defining what “best” entails in the context of their algorithm. Running detection with this policy should produce at most one detection.

JaniceEnrollmentType

Often times, the templates produced by algorithms will require different data for different use cases. The enrollment type indicates what the use case for the created template will be, allowing implementors to specialize their templates if they so desire. The use cases supported by the API are:

Role	Description
Janice11Reference	The template will be used as a reference template for 1:1 verification.
Janice11Verification	The template will be used for verification against a reference template in 1:1 verification.
Janice1NProbe	The template will be used as a probe template in 1:N search.
Janice1NGallery	The template will be enrolled into a gallery and searched against in 1:N search.
JaniceCluster	The template will be used for clustering.

Structs

JaniceContextType

A context object that manages hyperparameters for JanICE functions.

Minimum Object Size

This function specifies a minimum object size as one of its parameters. This value indicates the minimum size of objects that the user would like to see detected. Often, increasing the minimum size can improve runtime of algorithms. The size is in pixels and corresponds to the length of the smaller side of the rectangle. This means a detection will be returned if and only if its smaller side is larger than the value specified. If the user does not wish to specify a minimum width 0 can be provided.

Hint

Clustering is generally considered to be an ill-defined problem, and most algorithms require some help determining the appropriate number of clusters. The hint parameter helps influence the number of clusters, though the implementation is free to ignore it. The goal of the hint is to provide user input for two use cases:

1. If the hint is between 0 - 1 it should be regarded as a purity requirement for the algorithm. A 1 indicates the user wants perfectly pure clusters, even if that means more clusters are returned. A 0 indicates that the user wants very few clusters returned and accepts there may be some errors.
2. If the hint is > 1 it represents an estimated upper bound on the number of object types in the set.

Fields

Name	Type	Description
policy	JaniceDetectionPolicy	The detection policy
min_object_size	uint32_t	The minumum object size of a detection. See Minimum Object Size for additional information
role	JaniceEnrollmentType	The enrollment type for a template
threshold	double	The minimum acceptable score for a search result.
max_returns	uint32_t	The maximum number of results a single search should return
hint	double	A hint to a clustering algorithm. See Hint for additional information

JaniceContext

A pointer to a JaniceContextType.

Signature

```
typedef JaniceContextType* JaniceContext;
```

Functions

janice_create_context

Create a context object using the specified hyperparameters.

Signature

```
JANICE_EXPORT JaniceError janice_create_context(JaniceDetectionPolicy policy,
                                                uint32_t min_object_size,
                                                JaniceEnrollmentType role,
                                                double threshold,
                                                uint32_t max_returns,
                                                double hint,
                                                JaniceContext* context);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description

policy	JaniceDetectionPolicy	The detection policy
min_object_size	uint32_t	The minumum object size of a detection. See Minimum Object Size for additional information
role	JaniceEnrollmentType	The enrollment type for a template
threshold	double	The minimum acceptable score for a search result.
max_returns	uint32_t	The maximum number of results a single search should return
hint	double	A hint to a clustering algorithm. See Hint for additional information

janice_free_context

Free a JaniceContext object.

Signature

```
JANICE_EXPORT JaniceError janice_free_context(JaniceContext* context);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
context	JaniceContext*	The context object to free

Training

Functions

janice_fine_tune

Fine tune an implementation using new data. This function can be used to adapt an algorithm to a new domain. It is optional and can return *JANICE_NOT_IMPLEMENTED*. Artifacts created from fine tuning should be stored on disk and will be loaded in a successive initialization of the API.

Signature

```
JANICE_EXPORT JaniceError janice_fine_tune(JaniceMediaIterators media,
                                           JaniceTracksGroup group,
                                           int** labels,
                                           const char* output_prefix);
```

Thread Safety

This function is Thread Unsafe.

Parameters

Name	Type	Description
media	JaniceMediaIterators	A list of media objects to fine tune with.
tracks	JaniceTracksGroup	A collection of location information for objects in the fine tuning data. There must be the same number of sublists in this structure as there are elements in *media*. The tracks in the *ith* sublist of this structure give locations in the *ith* media object.
labels	int**	A list of lists of labels for objects in the fine tuning data. *labels[i][j]* should give the label for the *jth* track in the *ith* sublist of *tracks*.
output_prefix	const char*	A path to an existing directory with write access for the application. After successful fine tuning, this directory should be populated with all files necessary to initialize the API. Future calls to the API can use the fine tuned algorithm by passing *output_prefix* as the *sdk_path* parameter in janice_initialize.

Detection

Overview

In the context of this API, detection is used to refer to the identification of objects of interest within a I/O object. Detections are represented using the JaniceDetectionType object which implementors are free to define however they would like. For images, a detection is defined as a rectangle that bounds an object of interest and an associated confidence value. For video, a single object can exist in multiple frames. A rectangle and confidence are only relevant in a single frame. In this case, we define a detection as a list of (rectangle, confidence) pairs that track a single object through a video. It is not required that this list be dense however (i.e. frames can be skipped). To support this, we extend our representation of a detection to a (rectangle, confidence, frame) tuple where frame gives the index of the frame the rectangle was found in.

Structs

JaniceRect

A simple struct that represents a rectangle

Fields

Name	Type	Description
x	uint32_t	The x offset of the rectangle in pixels
y	uint32_t	The y offset of the rectangle in pixels
width	uint32_t	The width of the rectangle in pixels
height	uint32_t	The height of the rectangle in pixels

JaniceTrack

A structure to represent a track through a JaniceMediaIterator object. Tracks may be sparse (i.e. frames do not need to be sequential). Tracks are meant to follow a single object or area of interest, for example a face through multiple frames of a video.

Confidence

The confidence value indicates a likelihood that the rectangle actually bounds an object of interest. It is **NOT** required to be a probability and often only has meaning relative to other confidence values from the same algorithm. The only restriction is that a larger confidence value indicates a greater likelihood that the rectangle bounds an object.

Fields

Name	Type	Description
rects	JaniceRect*	A list of rectangles surrounding areas of interest in the media. This list should be *length* elements.
confidences	float*	A confidence to associate with each rectangle in *rects*. See Confidence for details about confidence values in this API. This list should be *length* elements.
frames	uint32_t*	The frame indices associated with each rectangle in *rects*. A track may be sparse and the indicies in this list are required to be sequential. This list should be *length* elements.
length	size_t	The number of rectangles, confidences, and frames in this structure.

JaniceTracks

A structure representing a list of JaniceTrack objects.

Fields

Name	Type	Description
tracks	JaniceTrack*	An array of track objects
length	size_t	The number of elements in *tracks*

JaniceTracksGroup

A structure representing a list of JaniceTracks objects.

Fields

Name	Type	Description
group	JaniceTracks*	An array of tracks objects
length	size_t	The number of elements in *group*

JaniceBuffer

An array of uint8_t

Signature

```
typedef uint8_t* JaniceBuffer;
```

JaniceDetectionType

An opaque pointer to a struct that represents a detection. See Detection for more information.

JaniceDetection

A pointer to a JaniceDetectionType object.

Signature

```
typedef struct JaniceDetectionType* JaniceDetection;
```

JaniceDetections

A structure to represent a list of JaniceDetection objects.

Fields

Name	Type	Description
detections	JaniceDetection*	An array of detection objects.
length	size_t	The number of elements in *detections*

JaniceDetectionsGroup

A structure to represent a list of JaniceDetections objects.

Fields

Name	Type	Description
group	JaniceDetections	An array of detections objects.
length	size_t	The number of elements in *group*

JaniceAttribute

A null-terminated string with an implementation defined format representing an attribute or a detection, template or gallery object. Implementations are free to define and implement attributes of their choice. For example, with face recognition an attribute might be:

- Gender
- Age
- Ethnicity
- Glasses
- etc.

Signature

```
typedef char* JaniceAttribute;
```

Functions

janice_create_detection_from_rect

Create a detection from a known rectangle. This is useful if a human has identified an object of interest and would like to run subsequent API functions on it. In the case where the input media is a video the given rectangle is considered an initial sighting of an object or region of interest. The implementation may detect additional sightings of the object in successive frames.

Signature

```
JANICE_EXPORT JaniceError janice_create_detection_from_rect(JaniceMediaIterator media,
                                                            const JaniceRect rect,
                                                            uint32_t frame,
                                                            JaniceDetection* detection);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
media	JaniceMediaIterator	A media object to create the detection from.
rect	const JaniceRect	A rectangle that bounds the object of interest.
frame	uint32_t	An index to the frame in the media where the object of interest appears. If the media is an image this should be 0.
detection	JaniceDetection*	An uninitialized pointer to a detection object. The object should allocated by the implementor during function execution. The user is responsible for freeing the object using janice_free_detection.

Example

```
JaniceMedia media; // Where media is a valid media object created previously

JaniceRect rect; // Create a bounding rectangle around an object of interest
rect.x      = 10; // The rectangle should fall within the bounds of the media
rect.y      = 10; // This code assumes media width > 110 and media height > 110
rect.width  = 100;
rect.height = 100;

JaniceDetection detection = NULL; // best practice to initialize to NULL
if (janice_create_detection(media, rect, 0 /* frame */, &detection) != JANICE_SUCCESS)
    // ERROR!
```

janice_create_detection_from_track

Create a detection from a known track. This is useful if a human has identified an object of interest and would like to run subsequent API functions on it.

Signature

```
JANICE_EXPORT JaniceError janice_create_detection_from_track(JaniceMediaIterator media,
                                                             const JaniceTrack track,
                                                             JaniceDetection* detection);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
media	JaniceMediaIterator	A media object to create the detection from.
track	JaniceTrack	A track bounding a region of through 1 or more frames.
detection	JaniceDetection*	An uninitialized pointer to a detection object. The object should allocated by the implementor during function execution. The user is responsible for freeing the object by calling janice_free_detection.

janice_detect

Automatically detect objects in a media object. See Detection for an overview of detection in the context of this API.

Signature

```
JANICE_EXPORT JaniceError janice_detect(JaniceMediaIterator media,
                                        JaniceContext context,
                                        JaniceDetections* detections);
```

Thread Safety

This function is Reentrant.

Tracking

When the input media is a video, implementations may implement a form of object tracking to correlate multiple sightings of the same object into a single structure. There are a number of approaches and algorithms to implement object tracking. This API makes NO attempt to define or otherwise constrain how implementations handle tracking. Users should be warned that an implementation might output multiple tracks for a single object and that a single track might contain multiple objects in it by mistake. In some cases, which should be clearly documented in implementation documentation, it might be beneficial to perform a post-processing clustering step on the results tracks, which could help correlate multiple tracks of the same object.

Parameters

Name	Type	Description

media	JaniceMediaIterator	A media object to run detection on.
context	JaniceContext	A context object with relevant hyperparameters set.
detections	JaniceDetections*	A struct to hold the resulting detections. Internal struct members should be initialized by the implementor as part of the call. The user is required to clear the struct by calling janice_clear_detections

Example

```
JaniceContext context = nullptr;
if (janice_create_context(JaniceDetectAll, // detection policy
                          24, // min_object_size, only find objects where the smaller side >
                          Janice1NProbe, // enrollment type, this shouldn't impact detection
                          0, // threshold, this shouldn't impact detection
                          0, // max_returns, this shouldn't impact detection
                          0, // hint, this shouldn't impact detection
                          &context) != JANICE_SUCCESS)
    // ERROR!

JaniceMedia media; // Where media is a valid media object created previously
JaniceDetections detections;
if (janice_detect(media, context, &detections) != JANICE_SUCCESS)
    // ERROR!
```

janice_detect_batch

Detect faces in a batch of media objects. Batch processing can often be more efficient than serial processing, particularly if a GPU or co-processor is being utilized.

Signature

```
JANICE_EXPORT JaniceError janice_detect_batch(JaniceMediaIterators media,
                                              JaniceContext context,
                                              JaniceDetectionsGroup* detections);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description

media	JaniceMediaIterators	An array of media iterators to run detection on.
context	JaniceContext	A context object with relevant hyperparameters set.
detections	JaniceDetectionsGroup*	A list of lists of detection objects. Each input media iterator can contain 0 or more possible detections. This output structure should mirror the input such that the sublist at index *i* should contain all of the detections found in media iterator *i*. If no detections are found in a particular media object an entry must still be present in the top-level output list and the sublist should have a length of 0. The implementor should allocate the internal members of this object during the call. The user is responsible for clearing the object by calling janice_clear_detections_group

janice_detection_get_track

Get a track object from a detection. The returned track should contain all rectangles, confidences, and frame indicies stored in the detection.

Signature

```
JANICE_EXPORT JaniceError janice_detection_get_track(JaniceDetection detection,
                                                     JaniceTrack* track);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
detection	JaniceDetection	The detection to get the track from.
track	JaniceTrack*	An uninitialized track object. This object should be allocated by the implementor during the call. The user is responsible for free this object by calling janice_clear_track.

janice_detection_get_attribute

Get an attribute from a detection. Attributes are additional metadata that an implementation might have when creating a detection. Examples from face detection include gender, ethnicity, and / or landmark locations.

Implementors are responsible for providing documentation on any attributes they support, valid key values and possible return values.

Signature

```
JANICE_EXPORT JaniceError janice_detection_get_attribute(JaniceDetection detection,
                                                         const char* key,
                                                         JaniceAttribute& value);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
detection	JaniceDetection	The detection object to extract the attribute from.
key	const char*	A key to look up a specific attribute. Valid keys must be defined and documented by the implementor.
value	JaniceAttribute*	An uninitialized char* to hold the value of the attribute. This object should be allocated by the implementor during the function call. The user is responsible for the object by calling janice_free_attribute.

janice_serialize_detection

Serialize a JaniceDetection object to a flat buffer.

Signature

```
JANICE_EXPORT JaniceError janice_serialize_detection(JaniceDetection detection,
                                                     JaniceBuffer* data,
                                                     size_t* len);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
detection	JaniceDetection	A detection object to serialize

data	JaniceBuffer*	An uninitialized buffer to hold the flattened data. The implementor should allocate this object during the function call. The user is required to free the object with janice_free_buffer.
len	size_t*	The length of the flat buffer after it is filled.

Example

```
JaniceDetection detection; // Where detection is a valid detection created
                           // previously.

JaniceBuffer buffer = NULL;
size_t buffer_len;
janice_serialize_detection(detection, &buffer, &buffer_len);
```

janice_deserialize_detection

Deserialize a JaniceDetection object from a flat buffer.

Signature

```
JANICE_EXPORT JaniceError janice_deserialize_detection(const JaniceBuffer data,
                                                       size_t len,
                                                       JaniceDetection* detection);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
data	const JaniceBuffer	A buffer containing data from a flattened detection object.
len	size_t	The length of the flat buffer.
detection	JaniceDetection*	An uninitialized detection object. This object should be allocated by the implementor during the function call. Users are required to free the object with janice_free_detection.

Example

```
const size_t buffer_len = K; // Where K is the known length of the buffer
JaniceBuffer buffer[buffer_len];

FILE* file = fopen("serialized.detection", "r");
fread(buffer, 1, buffer_len, file);

JaniceDetection detection = nullptr;
janice_deserialize_detection(buffer, buffer_len, detection);

fclose(file);
```

janice_read_detection

Read a detection from a file on disk. This method is functionally equivalent to the following-

```
const size_t buffer_len = K; // Where K is the known length of the buffer
JaniceBuffer buffer[buffer_len];

FILE* file = fopen("serialized.detection", "r");
fread(buffer, 1, buffer_len, file);

JaniceDetection detection = nullptr;
janice_deserialize_detection(buffer, buffer_len, detection);

fclose(file);
```

It is provided for memory efficiency and ease of use when reading from disk.

Signature

```
JANICE_EXPORT JaniceError janice_read_detection(const char* filename,
                                                JaniceDetection* detection);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
filename	const char*	The path to a file on disk
detection	JaniceDetection*	An uninitialized detection object.

Example

```
JaniceDetection detection = NULL;
if (janice_read_detection("example.detection", &detection) != JANICE_SUCCESS)
    // ERROR!
```

janice_write_detection

Write a detection to a file on disk. This method is functionally equivalent to the following-

```
JaniceDetection detection; // Where detection is a valid detection created
                           // previously.

JaniceBuffer buffer = NULL;
size_t buffer_len;
janice_serialize_detection(detection, &buffer, &buffer_len);

FILE* file = fopen("serialized.detection", "w+");
fwrite(buffer, 1, buffer_len, file);

fclose(file);
```

It is provided for memory efficiency and ease of use when writing to disk.

Signature

```
JANICE_EXPORT JaniceError janice_write_detection(JaniceDetection detection,
                                                 const char* filename);
```

ThreadSafety

This function is Reentrant.

Parameters

Name	Type	Description
detection	JaniceDetection	The detection object to write to disk.
filename	const char*	The path to a file on disk

Example

```
JaniceDetection detection; // Where detection is a valid detection created
                           // previously
if (janice_write_detection(detection, "example.detection") != JANICE_SUCCESS)
    // ERROR!
```

janice_free_buffer

Release the memory for an allocated buffer.

Signature

```
JANICE_EXPORT JaniceError janice_free_buffer(JaniceBuffer* buffer);
```

Thread Safety

This function is Reentrant

Parameters

Name	Type	Description
buffer	JaniceBuffer*	The buffer to free

janice_free_detection

Free any memory associated with a JaniceDetection object.

Signature

```
JANICE_EXPORT JaniceError janice_free_detection(JaniceDetection* detection);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
detection	JaniceDetection*	A detection object to free.

Example

```
JaniceDetection detection; // Where detection is a valid detection object
                           // created previously
```

```
if (janice_free_detection(&detection) != JANICE_SUCCESS)
    // ERROR!
```

janice_clear_detections

Free any memory associated with a JaniceDetections object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_detections(JaniceDetections* detections);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
detections	JaniceDetections *	A detection object to clear.

janice_clear_detections_group

Free any memory associated with a JaniceDetectionsGroup object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_detections_group(JaniceDetectionsGroup\* group);
```

janice_clear_track

Free any memory associated with a JaniceTrack object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_track(JaniceTrack* track);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
track	JaniceTrack	The track object to clear.

janice_clear_tracks

Free any memory associated with a of JaniceTracks object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_tracks(JaniceTracks* tracks);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
tracks	JaniceTracks*	A tracks objects to clear.

janice_clear_tracks_group

Free any memory associated with a JaniceTracksGroup object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_tracks_group(JaniceTracksGroup* group);
```

Parameters

Name	Type	Description
group	JaniceTracksGroup*	A tracks group to clear.

janice_free_attribute

Free any memory associated with an attribute value.

Signature

```
JANICE_EXPORT JaniceError janice_free_attribute(JaniceAttribute* value);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
attribute	JaniceAttribute*	The attribute to free.

Enrollment

Overview

This API defines feature extraction as the process of turning 1 or more Detection API objects that refer to the same object of interest into a single representation. This representation is defined in the API using the JaniceTemplateType object. In some cases (e.g. face recognition) this model of [multiple detections] -> [single representation] contradicts the current paradigm of [single detection] -> [single representation]. Implementors are free to implement whatever paradigm they choose internally (i.e. a JanICE template could be a simple list of single detection templates) provided the Comparison / Search functions work appropriately.

Failure To Enroll

For computer vision use cases, it is common to implement quality checks that can cause a template to fail during enrollment if it is missing certain characteristics. In this API templates should fail to enroll (FTE) quietly. This means that successive operations using an FTE template should still work without error. For example, calling janice_verify with an FTE template and a successful template should still return a score, even if that score is a predetermined

constant value like **-FLOAT_MAX**. Users can query a template to see if it failed to enroll using the janice_template_is_fte function and may choose to manually discard it if they desire.

Structs

JaniceTemplateType

An opaque pointer to a struct that represents a template.

Typedefs

JaniceTemplate

A pointer to a JaniceTemplateType object.

Signature

```
typedef struct JaniceTemplateType* JaniceTemplate;
```

JaniceTemplates

A structure representing a list of JaniceTemplate objects.

Fields

Name	Type	Description
tmpls	JaniceTemplate*	An array of template objects.
length	size_t	The number of elements in *tmpls*.

JaniceTemplatesGroup

A structure representing a list of JaniceTemplates objects.

Fields

Name	Type	Description
group	JaniceTemplates*	An array of templates objects.
length	size_t	The number of elements in *group*.

Functions

janice_enroll_from_media

Detect and enroll templates from a single media file. Detection should respect the provided minimum object size and detection policy. This function may produce 0 or more templates, depending on the number of objects found in the media.

Signature

```
JANICE_EXPORT JaniceError janice_enroll_from_media(JaniceMediaIterator media,
                                                   JaniceContext context,
                                                   JaniceTemplates* tmpls,
                                                   JaniceTracks* tracks);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
media	JaniceMediaIterator	The media to detect and enroll templates from.
context	JaniceContext	A context object with relevant hyperparameters set.
tmpls	JaniceTemplates*	A struct to hold the templates enrolled from the media. The internal members of this object should be allocated by the implementor during the call. The user is required to clear this object by calling janice_clear_templates
tracks	JaniceTracks*	A struct to hold the detection information for each of the templates enrolled from the media. This object should have the same number of elements as *tmpls*. The internal members of this object should be allocated by the implementor during the call. The user is required to clear this object by calling janice_clear_tracks.

janice_enroll_from_media_batch

Detect and enroll templates from a batch of media objects. Batch processing can often be more efficient then serial processing of a collection of data, particularly if a GPU or co-processor is being utilized.

Signature

```
JANICE_EXPORT JaniceError janice_enroll_from_media_batch(JaniceMediaIterators media,
                                                         JaniceContext context,
                                                         JaniceTemplatesGroup* tmpls,
                                                         JaniceTracksGroup* tracks);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description

media	JaniceMediaIterators	An array of media iterators to enroll.
context	JaniceContext	A context object with relevant hyperparameters set.
tmpls	JaniceTemplatesGroup*	A list of lists of template objects. Each input media iterator can contain 0 or more possible templates. This output structure should mirror the input such that the sublist at index *i* should contain all of the templates enrolled from media iterator *i*. If no templates are enrolled from a particular media object an entry must still be present in the top-level output list and the sublist should have a length of 0. The implementor should allocate the internal members of this object during the call. The user is responsible for clearing the object by calling janice_clear_templates_group.
tracks	JaniceTracksGroup*	A list of lists of track objects. The top level list should have the same number of elements as *tmpls* and sublist *i* should have the same number of elements as *tmpls* sublist i. Each track in the sublist should provide the location information for where the corresponding template was enrolled from. The implementor should allocate the internal members of this object during the call. The user is responsible for clearing the object by calling janice_clear_tracks_group.

janice_enroll_from_detections

Create a JaniceTemplate object from an array of detections.

Signature

```
JANICE_EXPORT JaniceError janice_enroll_from_detections(JaniceDetections detections,
                                                        JaniceContext context,
                                                        JaniceTemplate* tmpl);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description

detections	JaniceDetections	An array of detection objects.
context	JaniceContext	A context object with relevant hyperparameters set.
tmpl	JaniceTemplate*	An uninitialized template object. The implementor should allocate this object during the function call. The user is responsible for freeing the object by calling janice_free_template.

janice_enroll_from_detections_batch

Create a set of JaniceTemplate objects from an array of detections. Batch processing can often be more efficient then serial processing of a collection of data, particularly if a GPU or co-processor is being utilized.

Signature

```
JANICE_EXPORT JaniceError janice_enroll_from_detections_batch(JaniceDetectionsGroup detectio
                                                              JaniceContext context,
                                                              JaniceTemplates* tmpls);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
detections	Janice DetectionsGroup	A list of lists of detection objects. Multiple detections can be enrolled into a single template, for example if detections correspond to multiple views of the object of interest. Each sublist in this object should contain all detections that should be enrolled into a single template.
context	Janice Context	A context object with relevant hyperparameters set.
tmpls	Janice Templates*	A structure to hold the enrolled templates. This should have the same number of elements as *detections*. The implementor should allocate the internal members of this object during the call. The user is responsible for clearing the object by calling janice_clear_templates.

janice_template_is_fte

Query to see if a template has failed to enroll. See Failure To Enroll for additional information.

Signature

```
JANICE_EXPORT JaniceError janice_template_is_fte(JaniceTemplate tmpl,
                                                 int* fte);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
tmpl	JaniceTemplate	The template object to query.
fte	int*	FTE flag. If the template has not failed to enroll this should equal 0.

janice_template_get_attribute

Get a metadata value from a template using a key string. The valid set of keys is determined by the implementation and must be included in their delivered documentation. The possible return values for a valid key are also implementation specific. Invalid keys should return an error.

Signature

```
JANICE_EXPORT JaniceError janice_template_get_attribute(JaniceTemplate tmpl,
                                                        const char* key,
                                                        JaniceAttribute* value);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
tmpl	JaniceTemplate	A template object to query the attribute from.
key	const char*	A key to look up a specific attribute. Valid keys must be defined and documented by the implementor.
value	JaniceAttribute*	An uninitialized char* to hold the value of the attribute. This object should be allocated by the implementor during the function call. The user is responsible for the object by calling janice_free_attribute.

janice_serialize_template

Serialize a JaniceTemplate object to a flat buffer.

Signature

```
JANICE_EXPORT JaniceError janice_serialize_template(JaniceTemplate tmpl,
                                                    JaniceBuffer* data,
                                                    size_t* len);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
tmpl	JaniceTemplate	A template object to serialize
data	JaniceBuffer*	An uninitialized buffer to hold the flattened data. The implementor should allocate this object during the function call. The user is responsible for freeing the object by calling janice_free_buffer
len	size_t*	The length of the flat buffer.

Example

```
JaniceTemplate tmpl; // Where tmpl is a valid template created
                     // previously.

JaniceBuffer buffer = NULL;
size_t buffer_len;
janice_serialize_template(tmpl, &buffer, &buffer_len);
```

janice_deserialize_template

Deserialize a JaniceTemplate object from a flat buffer.

Signature

```
JANICE_EXPORT JaniceError janice_deserialize_template(const JaniceBuffer data,
                                                      size_t len,
                                                      JaniceTemplate* tmpl);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description

data	const JanuceBuffer	A buffer containing data from a flattened template object.
len	size_t	The length of the flat buffer.
tmpl	JaniceTemplate*	An uninitialized template object. The implementor should allocate this object during the function call. The user is responsible for freeing the object by calling janice_free_template.

Example

```
const size_t buffer_len = K; // Where K is the known length of the buffer
JaniceBuffer buffer[buffer_len];

FILE* file = fopen("serialized.template", "r");
fread(buffer, 1, buffer_len, file);

JaniceTemplate tmpl = NULL; // best practice to initialize to NULL
janice_deserialize_template(buffer, buffer_len, tmpl);

fclose(file);
```

janice_read_template

Read a template from a file on disk. This method is functionally equivalent to the following-

```
const size_t buffer_len = K; // Where K is the known length of the buffer
JaniceBuffer buffer[buffer_len];

FILE* file = fopen("serialized.template", "r");
fread(buffer, 1, buffer_len, file);

JaniceTemplate tmpl = nullptr;
janice_deserialize_template(buffer, buffer_len, tmpl);

fclose(file);
```

It is provided for memory efficiency and ease of use when reading from disk.

Signature

```
JANICE_EXPORT JaniceError janice_read_template(const char* filename,
                                               JaniceTemplate* tmpl);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description

filename	const char*	The path to a file on disk
tmpl	JaniceTemplate*	An uninitialized template object. The implementor should allocate this object during the function call. The user is responsible for freeing the object by calling janice_free_template.

Example

```
JaniceTemplate tmpl = NULL;
if (janice_read_template("example.template", &tmpl) != JANICE_SUCCESS)
    // ERROR!
```

janice_write_template

Write a template to a file on disk. This method is functionally equivalent to the following-

```
JaniceTemplate tmpl; // Where tmpl is a valid template created
                     // previously.

JaniceBuffer buffer = NULL;
size_t buffer_len;
janice_serialize_template(tmpl, &buffer, &buffer_len);

FILE* file = fopen("serialized.template", "w+");
fwrite(buffer, 1, buffer_len, file);

fclose(file);
```

It is provided for memory efficiency and ease of use when writing to disk.

Signature

```
JANICE_EXPORT JaniceError janice_write_template(JaniceTemplate tmpl,
                                                const char* filename);
```

ThreadSafety

This function is Reentrant.

Parameters

Name	Type	Description
tmpl	JaniceTemplate	The template object to write to disk.
filename	const char*	The path to a file on disk.

Example

```
JaniceTemplate tmpl; // Where tmpl is a valid template created
                     // previously
if (janice_write_template(tmpl, "example.template") != JANICE_SUCCESS)
    // ERROR!
```

janice_free_template

Free any memory associated with a JaniceTemplate object.

Signature

```
JANICE_EXPORT JaniceError janice_free_template(JaniceTemplate* tmpl);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
tmpl	JaniceTemplate	A template object to free.

Example

```
JaniceTemplate tmpl; // Where tmpl is a valid template object created previously
if (janice_free_template(&tmpl) != JANICE_SUCCESS)
    // ERROR!
```

janice_clear_templates

Free any memory associated with a of JaniceTemplates object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_templates(JaniceTemplates* templates);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
tmpls	JaniceTemplates*	A templates objects to clear.

janice_clear_templates_group

Free any memory associated with a JaniceTemplatesGroup object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_templates_group(JaniceTemplatesGroup* group);
```

Parameters

Name	Type	Description
group	JaniceTemplatesGroup*	A templates group to clear.

Gallery

Overview

This API defines a gallery object that represents a collection of templates. Galleries are useful in the 1-N use case (see Comparison / Search) when a user would like to query an unknown probe template against a set of known identities. A naive implementation of a gallery might be a simple array of templates. Often however, implementations have optimized algorithms or data structures that can lead to more efficient search times. It is recommended that advanced data structures be implemented as part of a gallery. Please note however the rules on gallery modification:

1. Gallery objects may be modified (templates inserted or removed) at any time.
2. It is understood that some preprocessing might need to be done between gallery modification and efficient search. A function janice_gallery_prepare exists for this purpose. The calling of this function is **OPTIONAL**. Please see janice_gallery_prepare for more information.

Structs

JaniceGalleryType

An opaque pointer to a struct that represents a gallery.

Typedefs

JaniceGallery

A pointer to a JaniceGalleryType object.

Signature

```
typedef struct JaniceGalleryType* JaniceGallery;
```

JaniceTemplateId

A unique identifier for a template.

Signature

```
typedef size_t JaniceTemplateId;
```

JaniceTemplateIds

A structure representing a list of JaniceTemplateId objects.

Fields

Name	Type	Description
ids	JaniceTemplateId*	An array of template id objects
length	size_t	The number of elements in *ids*

JaniceTemplateIdsGroup

A structure representing a list of JaniceTemplateIds objects.

Fields

Name	Type	Description
group	JaniceTemplateIds*	An array of template ids objects.

length	size_t	The number of elements in *group*

Functions

janice_create_gallery

Create a JaniceGallery object from a list of templates and unique ids.

Signature

```
JANICE_EXPORT JaniceError janice_create_gallery(JaniceTemplates tmpls,
                                                JaniceTemplateIds ids,
                                                JaniceGallery* gallery);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
tmpls	JaniceTemplates	An array of templates to add to the gallery. This can be *NULL* which would create an empty gallery. Data should be copied into the gallery. It is valid to pass an array with length 0 into this function, in which case an empty gallery should be initialized. This structure must have the same number of elements as *ids*.
ids	JaniceTemplateIds	A set of unique indentifiers to associate with the templates in *tmpls*. The *ith* id in this array corresponds to the *ith* input template. This structure must have the same number of elements as *tmpls*.
gallery	JaniceGallery*	An uninitialized gallery object. The implementor should allocate this object during the function call. The user is required to free this object by calling janice_free_gallery.

Example

```
JaniceTemplates tmpls; // Where tmpls is a valid array of valid template
                       // objects created previously
JaniceTemplateIds ids; // Where ids is a valid array of unique unsigned integers that
                       // is the same length as tmpls
JaniceGallery gallery = NULL; // best practice to initialize to NULL

if (janice_create_gallery(tmpls, ids, &gallery) != JANICE_SUCCESS)
    // ERROR!
```

janice_gallery_reserve

Reserve space in a gallery for N templates. This can save repeated allocations when doing multiple iterative inserts.

Signature

```
JANICE_EXPORT JaniceError janice_gallery_reserve(JaniceGallery gallery,
                                                 size_t n);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
gallery	JaniceGallery	The gallery to reserve space in.
n	size_t	The number of templates to reserve space for.

janice_gallery_insert

Insert a template into a gallery object. The template data should be copied into the gallery as the template may be deleted after this function.

Signature

```
JANICE_EXPORT JaniceError janice_gallery_insert(JaniceGallery gallery,
                                                JaniceTemplate tmpl,
                                                JaniceTemplateId id);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
gallery	JaniceGallery	A gallery object to insert the template into.
tmpl	JaniceTemplate	A template object to insert into the gallery. The template was created with the *Janice1NGallery* role. The template should be copied into the gallery. This object must remain in a valid state after this function call.
id	JaniceTemplateId	A unique id to associate with the input template. If the id is not unique the implementor should return *JANICE_DUPLICATE_ID*.

Example

```
JaniceTemplate tmpl; // Where tmpl is a valid template object created
                     // previously
JaniceTemplateId id; // Where id is a unique integer to associate with tmpl. This
                     // integer should not exist in the gallery
```

```
JaniceGallery gallery; // Where gallery is a valid gallery object created
                       // previously

if (janice_gallery_insert(gallery, tmpl, id) != JANICE_SUCCESS)
    // ERROR!
```

janice_gallery_insert_batch

Insert a batch of templates into a gallery.

Signature

```
JANICE_EXPORT JaniceError janice_gallery_insert_batch(JaniceGallery gallery,
                                                      JaniceTemplates tmpls,
                                                      JaniceTemplateIds ids);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
gallery	JaniceGallery	The gallery to insert the templates into.
tmpls	JaniceTemplates	The array of templates to insert in to the gallery. Each template was created with the *Janice1NGallery* role. Each template should be copied into the gallery by the implementor and must remain in a valid state after this function call. This structure must have the same number of elements as *ids*.
ids	JaniceTemplateIds	The array of unique ids to associate with *tmpls*. The *ith* id in this structure corresponds to the *ith* template in *tmpls*. This structure must have the same number of elements as *tmpls*.

janice_gallery_remove

Remove a template from a gallery object using its unique id.

Signature

```
JANICE_EXPORT JaniceError janice_gallery_remove(JaniceGallery gallery,
                                                JaniceTemplateId id);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
gallery	JaniceGallery	The gallery object to remove a template from.
id	JaniceTemplateId	The unique identifier for the template to remove from the gallery. If no template with the given ID is found in the gallery this function should return *JANICE_MISSING_ID*.

Example

```
JaniceTemplate tmpl; // Where tmpl is a valid template object created
                     // previously
JaniceTemplateId id = 0; // A unique integer id to associate with tmpl.

JaniceGallery gallery; // Where gallery is a valid gallery object created
                       // previously that does not have a template with id '0'
                       // already inserted in it.

// Insert the template with id 0
if (janice_gallery_insert(gallery, tmpl, id) != JANICE_SUCCESS)
    // ERROR!

// Now we can remove the template
if (janice_gallery_remove(gallery, id) != JANICE_SUCCESS)
    // ERROR!
```

janice_gallery_remove_batch

Remove a batch of templates from a gallery.

Signature

```
JANICE_EXPORT JaniceError janice_gallery_remove_batch(JaniceGallery gallery,
                                                      JaniceTemplateIds ids);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
gallery	JaniceGallery	The gallery object to remove the templates from.
ids	JaniceTemplateIds	The unique identifiers for the templates to remove from the gallery.

janice_gallery_prepare

Prepare a gallery for search. Implementors can use this function as an opportunity to streamline gallery objects to accelerate the search process. The calling convention for this function is **NOT** specified by the API, this means that this function is not guaranteed to be called before janice_search. It also means that templates can be added to a gallery before and after this function is called. Implementations should handle all of these calling conventions. However, users should be aware that this function may be computationally expensive. They should strive to call it only at critical junctions before search and as few times as possible overall.

Signature

```
JANICE_EXPORT JaniceError janice_gallery_prepare(JaniceGallery gallery);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
gallery	JaniceGallery	A gallery object to prepare

Example

```
JaniceTemplate* tmpls; // Where tmpls is a valid array of valid template
                       // objects created previously
JaniceTemplateIds ids; // Where ids is a valid array of unique unsigned integers that
                       // is the same length as tmpls
JaniceTemplate tmpl; // Where tmpl is a valid template object created
                     // previously
JaniceTemplateId id; // Where id is a unique integer id to associate with tmpl.

JaniceGallery gallery = NULL; // best practice to initialize to NULL

if (janice_create_gallery(tmpls, ids, &gallery) != JANICE_SUCCESS)
    // ERROR!

// It is valid to run search without calling prepare
if (janice_search(tmpl, gallery ... ) != JANICE_SUCCESS)
    // ERROR!

// Prepare can be called after search
if (janice_gallery_prepare(gallery) != JANICE_SUCCESS)
    // ERROR!

// Search can be called again right after prepare
if (janice_search(tmpl, gallery ... ) != JANICE_SUCCESS)
    // ERROR!

// Insert another template into the gallery. This is valid after the gallery
// has been prepared
if (janice_gallery_insert(gallery, tmpl, 112) != JANICE_SUCCESS)
    // ERROR!

// Prepare the gallery again
if (janice_gallery_prepare(gallery) != JANICE_SUCCESS)
    // ERROR!
```

janice_serialize_gallery

Serialize a JaniceGallery object to a flat buffer.

Signature

```
JANICE_EXPORT JaniceError janice_serialize_gallery(JaniceConstGallery gallery,
                                                   JaniceBuffer* data,
                                                   size_t* len);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
gallery	Janice Gallery	A gallery object to serialize
data	JaniceBuffer	An uninitialized buffer to hold the flattened data. The implementor allocate this object during the function call. The user is responsible for freeing this object by calling janice_free_buffer.
len	size_t*	The length of the flat buffer after it is allocated.

Example

```
JaniceGallery gallery; // Where gallery is a valid gallery created
                       // previously.

JaniceBuffer buffer = NULL;
size_t buffer_len;
janice_serialize_gallery(gallery, &buffer, &buffer_len);
```

janice_deserialize_gallery

Deserialize a JaniceGallery object from a flat buffer.

Signature

```
JANICE_EXPORT JaniceError janice_deserialize_gallery(const JaniceBuffer data,
                                                     size_t len,
                                                     JaniceGallery* gallery);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
data	const JaniceBuffer	A buffer containing data from a flattened gallery object.
len	size_t	The length of the flat buffer.

gallery	JaniceGallery*	An uninitialized gallery object. The implementor should allocate this object during the function call. The user is responsible for freeing the object by calling janice_free_gallery.

Example

```
const size_t buffer_len = K; // Where K is the known length of the buffer
unsigned char buffer[buffer_len];

FILE* file = fopen("serialized.gallery", "r");
fread(buffer, 1, buffer_len, file);

JaniceGallery gallery = NULL; // best practice to initialize to NULL
janice_deserialize_gallery(buffer, buffer_len, gallery);

fclose(file);
```

janice_read_gallery

Read a gallery from a file on disk. This method is functionally equivalent to the following-

```
const size_t buffer_len = K; // Where K is the known length of the buffer
JaniceBuffer buffer[buffer_len];

FILE* file = fopen("serialized.gallery", "r");
fread(buffer, 1, buffer_len, file);

JaniceGallery gallery = NULL; // best practice to initialize to NULL
janice_deserialize_gallery(buffer, buffer_len, gallery);

fclose(file);
```

It is provided for memory efficiency and ease of use when reading from disk.

Signature

```
JANICE_EXPORT JaniceError janice_read_gallery(const char* filename,
                                              JaniceGallery* gallery);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
filename	const char*	The path to a file on disk

gallery	JaniceGallery*	An uninitialized gallery object. The implementor should allocate this object during the function call. The user is responsible for freeing this object by calling janice_free_gallery.

Example

```
JaniceGallery gallery = NULL;
if (janice_read_gallery("example.gallery", &gallery) != JANICE_SUCCESS)
    // ERROR!
```

janice_write_gallery

Write a gallery to a file on disk. This method is functionally equivalent to the following-

```
JaniceGallery gallery; // Where gallery is a valid gallery created previously.

JaniceBuffer buffer = NULL;
size_t buffer_len;
janice_serialize_gallery(gallery, &buffer, &buffer_len);

FILE* file = fopen("serialized.gallery", "w+");
fwrite(buffer, 1, buffer_len, file);

fclose(file);
```

It is provided for memory efficiency and ease of use when writing to disk.

Signature

```
JANICE_EXPORT JaniceError janice_write_gallery(JaniceConstGallery gallery,
                                               const char* filename);
```

ThreadSafety

This function is Reentrant.

Parameters

Name	Type	Description
gallery	JaniceGallery	The gallery object to write to disk.
filename	const char*	The path to a file on disk

Example

```
JaniceGallery gallery; // Where gallery is a valid gallery created previously
if (janice_write_gallery(gallery, "example.gallery") != JANICE_SUCCESS)
    // ERROR!
```

janice_free_gallery

Free any memory associated with a JaniceGalleryType object.

Signature

```
JANICE_EXPORT JaniceError janice_free_gallery(JaniceGallery* gallery);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
gallery	JaniceGallery*	A gallery object to free.

Example

```
JaniceGallery gallery; // Where gallery is a valid gallery object created previously
if (janice_free_gallery(&gallery) != JANICE_SUCCESS)
    // ERROR!
```

janice_clear_template_ids

Free any memory associated with a of JaniceTemplateIds object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_template_ids(JaniceTemplateIds* ids);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
ids	JaniceTemplateIds*	A template ids objects to clear.

janice_clear_template_ids_group

Free any memory associated with a JaniceTemplateIdsGroup object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_template_ids_group(JaniceTemplateIdsGroup* group);
```

Parameters

Name	Type	Description
group	JaniceTemplateIdsGroup*	A template ids group to clear.

Comparison / Search

Overview

This API defines two possible types of comparisons, 1:1 and 1:N. These are represented by the janice_verify and janice_search functions respectively. The API quantifies the relationship between two templates as a single number called a Similarity Score.

Structs

JaniceSimilarity

A number representing the similarity between two templates. See Similarity Score for more information.

Signature

```
typedef double JaniceSimilarity
```

JaniceSimilarities

A structure representing a list of JaniceSimilarity objects.

Fields

Name	Type	Description
similarities	JaniceSimilarity*	An array of similarity objects.
length	size_t	The number of elements in *similarities*.

JaniceSimilaritiesGroup

A structure representing a list of JaniceSimilarities objects.

Fields

Name	Type	Description
group	JaniceSimilarities*	An array of similarities objects.
length	size_t	The number of elements in *group*.

Functions

janice_verify

Compare two templates with the difference expressed as a similarity score.

Signature

```
JANICE_EXPORT JaniceError janice_verify(JaniceConstTemplate reference,
                                        JaniceConstTemplate verification,
                                        JaniceSimilarity* similarity);
```

Thread Safety

This function is Reentrant.

Similarity Score

This API expects that the comparison of two templates results in a single value that quantifies the similarity between them. A similarity score is constrained by the following requirements:

1. Higher scores indicate greater similarity
2. Scores can be asymmetric. This mean verify(a, b) does not necessarily equal verify(b, a)

Parameters

Name	Type	Description

reference	JaniceTemplate	A reference template. This template was created with the *Janice11Reference* role.
verification	JaniceTemplate	A verification template. This this template was created with the *Janice11Verification* role.
similarity	JaniceSimilarity*	A similarity score. See Similarity Score.

Example

```
JaniceTemplate reference; // Where reference is a valid template object created
                          // previously
JaniceTemplate verification; // Where verification is a valid template object
                             // created previously
JaniceSimilarity similarity;
if (janice_verify(reference, verification, &similarity) != JANICE_SUCCESS)
    // ERROR!
```

janice_verify_batch

Compute a batch of reference templates with a batch of verification templates. The *ith* in the reference batch is compared with the *ith* template in the verification batch. Batch processing can often be more efficient than serial processing, particularly if a GPU or co-processor is being utilized.

Signature

```
JANICE_EXPORT JaniceError janice_verify_batch(JaniceTemplates references,
                                              JaniceTemplates verifications,
                                              JaniceSimilarities* similarities);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
references	JaniceTemplates	An array of reference templates. Each template was created with the *Janice11Reference* role.
verifications	JaniceTemplates	An array of verification templates. Each template was created with the *Janice11Verification* role. The number of elements in *verifications* must equal the number of elements in *references*.

similarities	JaniceSimilarities *	A struct to hold the output similarity scores. There must be the same number of similarity scores output as there are *references* and *verifications*. The implementor should allocate the internal members of this object during the call. The user is responsible for clearing the object by calling janice_clear_similarities.

janice_search

Compute 1-N search results between a query template object and a target gallery object. When running searches, users will often only want the top N results, or will only want results above a predefined threshold. This function must respect the *threshold* and *max_returns* fields of a JaniceContext object to facilitate these use cases. Implementors must always respect the passed threshold (i.e. a score below the given threshold should never be returned). If users would not like to specify a threshold they can set the member to **-DOUBLE_MAX**. If the *max_returns* member is non-zero implementors should respect both the threshold and the number of desired returns (i.e. return the top K scores above the given threshold). Users who would like to see all valid returns should set *max_returns* to 0.

This function allocates two structures with the same number of elements. *similarities* is a JaniceSimilarities object with an arra of Similarity Score, sorted in descending order. The second is a JaniceTemplateIds where the *ith* template id gives the unique identifier for the gallery template that produces the *ith* similarity score when compared with the probe.

Signature

```
JANICE_EXPORT JaniceError janice_search(JaniceConstTemplate probe,
                                       JaniceConstGallery gallery,
                                       JaniceContext context,
                                       JaniceSimilarities* similarities,
                                       JaniceTemplateIds* ids);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
probe	JaniceTemplate	A query template. The template was created with the *Janice1NProbe* role.
gallery	JaniceGallery	A gallery object to search against.

context	JaniceContext	A context object with relevant hyperparameters set.
similarities	JaniceSimilarities*	A structure to hold the output similarity scores, sorted in descending order. This structure should have the same number of elements as *ids*. The implementor should allocate the internal members of this object during the call. The user is responsible for clearing the object by calling janice_clear_similarities.
ids	JaniceTemplateIds*	A structure to hold the gallery template ids associated with the *similarities*. This structure should have the same number of elements as *similarities*. The implementor should allocate the internal members of this object during the call. The user is responsible for clearing the object by calling janice_clear_template_ids.

Example

```
JaniceTemplate probe;  // Where probe is a valid template object created
                       // previously
JaniceGallery gallery; // Where gallery is a valid gallery object created
                       // previously

JaniceContext context = nullptr;
if (janice_create_context(JaniceDetectAll, // detection policy, this shouldn't impact search
                          0, // min_object_size, this shouldn't impact search
                          Janice1NProbe, // enrollment type, this shouldn't impact search
                          0.7, // threshold, get all matches scoring above 0.7
                          50, // max_returns, get the top 50 matches scoring above the set t
                          0, // hint, this shouldn't impact search
                          &context) != JANICE_SUCCESS)
    // ERROR!

JaniceSimilarities similarities;
JaniceTemplateIds ids;

// Run search
if (janice_search(probe, gallery, context, &similarities, &ids) != JANICE_SUCCESS)
    // ERROR!
```

janice_search_batch

Compute 1-N search results between a batch of probe templates and a single gallery. Given *N* probe templates in a batch, this function should return a single JaniceSimilaritiesGroup with N sublists and a single JaniceTemplateIdsGroup with N sublists. Each sublist must conform to the behavior defined in janice_search. Batch processing can often be more efficient than serial processing, particularly if a GPU or co-processor is being utilized.

Signature

```
JANICE_EXPORT JaniceError janice_search_batch(JaniceTemplates probes,
                                              JaniceGallery gallery,
                                              JaniceContext context,
```

```
                                                    JaniceSimilaritiesGroup* similarities,
                                                    JaniceTemplateIdsGroup* ids);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
probes	Janice Templates	An array of probe templates to search with. Each template was created with the *Janice1NProbe* role.
gallery	Janice Gallery	The gallery to search against.
context	Janice Context	A context object with relevant hyperparameters set.
similarities	Janice SimilaritiesGroup*	A structure to hold the output similarities. Given *N* probes, there should be *N* sublists in the output, where the *ith* sublist gives the similarity scores of the *ith* probe. Internal struct members should be initialized by the implementor as part of the call. The user is required to clear the struct by calling janice_clear_similarities_group.
ids	Janice TemplateIds Group *	A structure to hold the output template ids. Given *N* probes, there should be *N* sublists in the output, where the *ith* sublist gives the gallery template ids of the *ith* probe. Internal struct members should be initialized by the implementor as part of the call. The user is required to clear the struct by calling janice_clear_template_ids_group.

janice_clear_similarities

Free any memory associated with a JaniceSimilarities object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_similarities(JaniceSimilarities* similarities);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
similarities	JaniceSimilarities*	An similarities object to clear.

janice_clear_similarities_group

Free any memory associated with a JaniceSimilaritiesGroup object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_similarities_group(JaniceSimilaritiesGroup* group);
```

Parameters

Name	Type	Description
group	JaniceSimilaritiesGroup*	A similarities group to clear.

Clustering

Overview

This API defines clustering is the automatic and unsupervised combination of unlabelled templates into groups of like templates. What constitutes likeness is heavily dependent on the use case and context in question. One example when dealing with faces is grouping based on identity, where all faces belonging to a single individual are placed in a cluster.

Structs

JaniceClusterId

A unique identifier for a cluster

Signature

```
typedef size_t JaniceClusterId;
```

JaniceClusterIds

A structure to represent a list of JaniceClusterId objects.

Fields

Name	Type	Description
ids	JaniceClusterId*	An array of cluster id objects.
length	size_t	The number of elements in *ids*

JaniceClusterIdsGroup

A structure to represent a list of JaniceClusterIds objects.

Fields

Name	Type	Description
group	JaniceClusterIds*	An array of cluster ids objects.
length	size_t	The number of elements in *group*

JaniceClusterConfidence

A value representing the confidence that an item belongs to a cluster.

Signature

```
typedef double JaniceClusterConfidence;
```

JaniceClusterConfidences

A structure to represent a list of JaniceClusterConfidence objects.

Fields

Name	Type	Description
confidences	JaniceClusterConfidence*	An array of cluster confidence objects.
length	size_t	The number of elements in *confidences*

JaniceClusterConfidencesGroup

A structure to represent a list of JaniceClusterConfidences objects.

Fields

Name	Type	Description
group	JaniceClusterConfidences*	An array of cluster confidences objects.
length	size_t	The number of elements in *group*

Function

janice_cluster_media

Cluster a collection of media objects into groups. Each media object may contain 0 or more objects of interest. The output is arranged so that each output structure has *N* sublists where *N* is the number of input media and the *ith* sublist contains information for objects found in the *ith* media.

Cluster Confidence

Along with a cluster assignment, this API supports the concept of a cluster confidence. A cluster confidence is a value indicating a liklihood that the object of interest actually belongs to a cluster. For example, one possible implementation of a cluster confidence is the negative distance of an object from the cluster centroid. One use case for this value, is for end users to manually scrub cluster results by dynamically orphaning elements with lower confidence values. The cluster confidence is subject to the following contraints:

1. A higher value indicates greater confidence of cluster membership
2. No meaning can be assigned to an individual confidence, it is only relevant when being compared with other confidences generated by the same algorithm.

Signature

```
JANICE_EXPORT JaniceError janice_cluster_media(JaniceMediaIterators media,
                                               JaniceContext context,
                                               JaniceClusterIdsGroup* cluster_ids,
                                               JaniceClusterConfidencesGroup* cluster_confid
                                               JaniceTracksGroup* tracks);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
media	JaniceMediaIterators	An array of media to cluster.
context	JaniceContext	A context object with relevant hyperparameters set.
cluster_ids	JaniceClusterIdsGroup*	An output structure to hold cluster ids. Objects with the same cluster id are members of the same cluster. This structure must have *N* sublists, where *N* is the number of elements in *media*. The *ith* sublist contains cluster ids for all objects of interest found in the *ith* media. If no objects of interest are found in a media then the corresponding sublist should have length 0. Internal struct members should be initialized by the implementor as part of the call. The user is required to clear the struct by calling janice_clear_cluster_ids_group.

cluster_confidences	JaniceClusterConfidencesGroup*	An output structure to hold Cluster Confidence. This structure must have *N* sublists, where *N* is the number of elements in *media*. The *ith* sublist contains cluster confidences for all objects of interest found in the *ith* media. The *jth* confidence in the *ith* sublist refers to the same object as the *jth* id in the *ith* sublist of *ids*. Internal struct members should be initialized by the implementor as part of the call. The user is required to clear the struct by calling janice_clear_cluster_confidences_group.
tracks	JaniceTracksGroup*	Location information for each clustered object. This structure must have *N* sublists, where *N* is the number of elements in *media*. The *ith* sublist contains tracks for all objects of interest found in the *ith* media. The *jth* track in the *ith* sublist refers to the same object as the *jth* id in the *ith* sublist of *ids* and the *jth* confidence in the *ith* sublist of confidences. Internal struct members should be initialized by the implementor as part of the call. The user is required to clear the struct by calling janice_clear_tracks_group.

janice_cluster_templates

Cluster a collection of template objects into groups.

Signature

```
JANICE_EXPORT JaniceError janice_cluster_templates(JaniceTemplates tmpls,
                                                   JaniceContext context,
                                                   JaniceClusterIds* cluster_ids,
                                                   JaniceClusterConfidences* cluster_confide
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
tmpls	JaniceTemplates	An array of templates to cluster. Each template was created with the *JaniceCluster* role.
context	JaniceContext	A context object with relevant hyperparameters set.

cluster_ids	JaniceClusterIds*	An output structure to hold cluster ids. Templates assigned the same cluster id are members of the same cluster. This structure must have the same number of elements as *tmpls*. The *ith* cluster id corresponds to the *ith* template object. Objects that can't be clustered should be assigned a unique cluster id. Internal struct members should be initialized by the implementor as part of the call. The user is required to clear the struct by calling janice_clear_cluster_ids.
cluster_confidences	JaniceClusterConfidences*	An output structure to hold Cluster Confidence. This structure must have the same number of elements as *tmpls*. The *ith* cluster confidence corresponds to the *ith* template object. Internal struct members should be initialized by the implementor as part of the call. The user is required to clear the struct by calling janice_clear_cluster_confidences.

janice_clear_cluster_ids

Free any memory associated with a of JaniceClusterIds object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_cluster_ids(JaniceClusterIds* ids);
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
ids	JaniceClusterIds*	A cluster ids object to clear.

janice_clear_cluster_ids_group

Free any memory associated with a JaniceClusterIdsGroup object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_cluster_ids_group(JaniceClusterIdsGroup* group);
```

Parameters

Name	Type	Description
group	JaniceClusterIdsGroup*	A cluster ids group to clear.

janice_clear_cluster_confidences

Free any memory associated with a of JaniceClusterConfidences object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_cluster_confidences(JaniceClusterConfidences* confide
```

Thread Safety

This function is Reentrant.

Parameters

Name	Type	Description
confidences	JaniceClusterConfidences*	A cluster confidences object to clear.

janice_clear_cluster_confidences_group

Free any memory associated with a JaniceClusterConfidencesGroup object.

Signature

```
JANICE_EXPORT JaniceError janice_clear_cluster_confidences_group(JaniceClusterConfidencesGro
```

Parameters

Name	Type	Description
group	JaniceClusterConfidencesGroup*	A cluster confidences group to clear.

License

The JanICE API is a *C* API that provides a common interface between computer vision algorithms and agencies and entities that would like to use them. The API consists of a core header file defining required *C* functions. It also defines a number of interfaces to other languages on top of the *C* API.

About

Computer vision is a rapidly expanding and improving field that has seen significant progress in it's capabilities over the past decade. Government agencies can leverage computer vision algorithms to better understand images and videos that they ingest. This in turn can lead to improved response times, increased public safety, and numerous other benefits. The JanICE API provides a common framework that commercial vendors and government agencies can use to ease integration between algorithms and use cases. The API aims to cover a number of different Computer Vision subproblems. At this time, these problems include:

- Face Recognition

Some function calls serve multiple use cases in different ways. In those cases the function documentation strives to clearly indicate the differences. If no differences are indicated it means that the function is universal in that it applies the same to each subproblem addressed by the API.

This work is being sponsored by The Department of Homeland Security; Science and Technology Directorate.

Focus Areas

Face Recognition

Facial recognition has emerged as a key technology for government agencies to efficiently triage and analyze large data streams. A large ecosystem of facial recognition algorithms already exists from a variety of sources including commercial vendors, government programs and academia. However, integrating this important technology into existing technology stacks is a difficult and expensive endeavor. The JanICE API aims to address this problem by functioning as a compatibility layer between users and the algorithms. Users can write their applications on "top" of the API while algorithm providers will implement their algorithms "beneath" the API. This means that users can write their applications independent of any single FR algorithm and gives them the freedom to select the algorithm or algorithms that best serve their specific use case without worrying about integration. Algorithm providers will be able to serve their algorithms across teams and agencies without having to integrate with the different tools and services of each specific team.

License

Indices and tables

- **genindex**
- **modindex**
- **search**