JanICE documentation

version 4.0.1

Noblis

August 10, 2017

Contents

Welcome to JanICE's documentation!

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

The API is provided under the MIT license(LICENSE.txt) and is *free for academic and commercial use*.

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:

Type	Description
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_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_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_ENROLL	Could not construct a template
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.

Miscellaneous

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 video respectively. These interfaces allow pixel-level access for implementations and can be changed independently to work with new formats.

Structs

JaniceImageType

An interface 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	JaniceBuffer	A contiguous, row-major array containing pixel data.
owner	bool	True if the image owns its data and should delete it, false otherwise.

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 iterators one frame. The next frame or video image should be stored in the JaniceImage parameter.
seek	JaniceError(JaniceMediaIteratorType *, uint32_t)	A function pointer that advances the iterator to a specific frame. This function is not applicable to images.
get	JaniceError(JaniceMediaIteratorType *, JaniceImage*, uint32_t)	A function pointer that advances the iterator to a specific frame and retrieves that frame. This function is not applicable to images.
tell	JaniceError(JaniceMediaIteratorType *, uint32_t *)	A function pointer to report the current position of the iterator. This function is not applicable to images.
free_i	JaniceError(JaniceImage *)	A function pointer to free an image allocated by *next* or *get*.
free	JaniceError(JaniceMediaIteratorType **)	A function pointer to free a JaniceMediaIteratorType object.

Typedefs

JaniceBuffer

An array of uint8_t

Signature

```
typedef uint8_t* JaniceBuffer;
```

JaniceImage

A pointer to a JaniceImageType object.

Signature

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

JaniceConstImage

A pointer to a constant JaniceImageType object.

Signature

```
typedef const struct JaniceImageType* JaniceConstImage;
```

JaniceMediaIterator

A pointer to a JaniceMediaIteratorType object.

Signature

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

JaniceMediaIterators

A pointer to an array of JaniceMediaIterator objects.

Signature

```
typedef struct JaniceMediaIterator* JaniceMediaIterators;
```

Functions

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_image_access

Get a pixel value at a given row, column and channel.

Signature

```
inline uint8_t janice_image_access(JaniceConstImage image, uint32_t channel, uint32_t row, u
```

Thread Safety

This function is reentrant

Parameters

Name	Type	Description
image	JaniceConstImage	An image object
channel	uint32_t	The channel to access. Must be less image->channels.
row	uint32_t	The row to access. Must be less than image->rows.
col	uint32_t	The column to access. Must be less than image->cols.

Training

Functions

janice_train

Train an implementation using new data.

Signature

```
JANICE_EXPORT JaniceError janice_train(const char* data_prefix,
                                       const char* data_list);
```

Thread Safety

This function is thread unsafe.

Parameters

Name	Type	Description
data_prefix	const char*	A prefix path pointing to the locations of the training data.
data_train	const char*	A list of training data and labels. The format is currently unspecified.

Notes

This function is untested, unstable and most likely subject to breaking changes in future releases.

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

JaniceDetectionIteratorType

An opaque pointer to an iterator class through a detection. If the detection was computed from an image, the iterator should only move over a single value. If the detection was computed from a video, the iterator should move over an array of elements, the length of which is less than or equal to the number of frames in the video, and might be sparse (i.e. frames can be skipped).

JaniceDetectionType

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

Typedefs

JaniceDetectionIterator

A pointer to a JaniceDetectionIteratorType object.

Signature

```
typedef struct JaniceDetectionIteratorType* JaniceDetectionIterator;
```

JaniceDetection

A pointer to a JaniceDetectionType object.

Signature

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

JaniceConstDetection

A pointer to a constant JaniceDetectionType object.

Signature

```
typedef const struct JaniceDetectionType* JaniceConstDetection;
```

JaniceDetections

An array of JaniceDetection objects.

Signature

```
typedef struct JaniceDetection* JaniceDetections;
```

JaniceConstDetections

An array of JaniceConstDetection objects.

Signature

```
typedef struct JaniceConstDetection* JaniceConstDetections;
```

Functions

janice_detection_it_next

Get the next element in a detection track.

Signature

```
JANICE_EXPORT JaniceError janice_detection_it_next(JaniceDetectionIterator it,
                                                   JaniceRect* rect,
                                                   uint32_t* frame,
                                                   float* confidence);
```

Thread Safety

This function is reentrant.

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.

Parameters

Name	Type	Description
it	JaniceDetectionIterator	A detection iterator object.
rect	JaniceRect *	The location of a object of interest.
frame	uint32_t *	The frame index for an object of interest.
confidence	float *	The Confidence of the location.

janice_detection_it_reset

Reset an iterator back to its initial state.

Signature

```
JANICE_EXPORT JaniceError janice_detection_it_reset(JaniceDetectionIterator it);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
it	JaniceDetectionIterator	The iterator object to reset.

janice_free_detection_it

Free any memory associated with a detection iterator object.

Signature

```
JANICE_EXPORT JaniceError janice_free_detection_it(JaniceDetectionIterator* it);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
it	JaniceDetectionIterator *	The iterator object to free.

janice_create_detection

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 the object of interest. The implementation may detect additional sightings of the object in successive frames.

Signature

```
JANICE_EXPORT JaniceError janice_create_detection(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_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,
                                        uint32_t min_object_size,
```

```
                                        JaniceDetections* detections,
                                        uint32_t* num_detections);
```

Thread Safety

This function is reentrant.

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.

Tracking

When the input media is a video many implementations will 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.
min_object_ size	uint32_t	A minimum object size. See Minimum Object Size
detections	JaniceDetections *	An uninitialized array to hold all of the detections detected in the media object. This object should be allocated by the implementor during the call. The user is required to free the object by calling janice_free_detections.
num_ detections	uint32_t*	The number of detections returned in the *detections* array.

Example

```
JaniceMedia media; // Where media is a valid media object created previously
const uint32_t min_object_size = 24; // Only find objects where the smaller
                                     // side is > 24 pixels
JaniceDetection* detections = NULL; // best practice to initialize to NULL
uint32_t num_detections; // Will be populated with the size of detections

if (janice_detect(media, min_object_size, &detections, &num_detections) != JANICE_SUCCESS)
    // ERROR!
```

janice_create_detection_it

Create an iterator to iterate over detection elements.

Signature

```
JANICE_EXPORT JaniceError janice_create_detection_it(JaniceConstDetection detection,
                                                     JaniceDetectionIterator* it);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
detection	JaniceConstDetection	The detection object to create an iterator from.
it	JaniceDetectionIterator *	An uninitialized detection iterator object. The implementor should allocate this object during the function call. Users are required to free the object with janice_free_detection_it .

janice_serialize_detection

Serialize a JaniceDetection object to a flat buffer.

Signature

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

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
detection	JaniceConstDetection	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(JaniceConstDetection detection,
                                                 const char* filename);
```

ThreadSafety

This function is reentrant.

Parameters

Name	Type	Description
detection	JaniceConstDetection	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_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. Best practice dictates the pointer should be set to *NULL* after it is freed.

Example

```
JaniceDetection detection; // Where detection is a valid detection object
                           // created previously
if (janice_free_detection(&detection) != JANICE_SUCCESS)
    // ERROR!
```

janice_free_detections

Free any memory associated with a:ref:*JaniceDetections* object.

Signature

```
JANICE_EXPORT JaniceError janice_free_detections(JaniceDetections* detection,
                                                 uint32_t num_detections);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
detections	JaniceDetections *	An array of detections to free. Best practice dictates the pointer should be set to *NULL* after it is freed.

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 functions work appropriately.

Enumerations

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:

Type	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

JaniceTemplateType

An opaque pointer to a struct that represents a template.

Typedefs

JaniceTemplate

A pointer to a JaniceTemplateType object.

Signature

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

JaniceConstTemplate

A pointer to a constant JaniceTemplateType object.

Signature

```
typedef const struct JaniceTemplateType* JaniceConstTemplate;
```

JaniceTemplates

An array of JaniceTemplate objects.

Signature

```
typedef struct JaniceTemplate* JaniceTemplates;
```

JaniceConstTemplates

An array of JaniceConstTemplate objects.

Signature

```
typedef struct JaniceConstTemplate* JaniceConstTemplates;
```

Functions

janice_create_template

Create a JaniceTemplate object from an array of detections.

Signature

```
JANICE_EXPORT JaniceError janice_create_template(JaniceConstDetections detections,
                                                 uint32_t num_detections,
                                                 JaniceEnrollmentType role,
                                                 JaniceTemplate* tmpl);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
detections	JaniceConstDetections	An array of detection objects.
num_detections	uint32_t	The number of input detections.
role	JaniceEnrollmentType	The use case for the template
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

```
JaniceDetections detections; // Where detections is a valid array of valid
                             // detection objects created previously
const uint32_t num_detections = K; // Where K is the number of detections in
                                   // the detections array
JaniceEnrollmentType role = Janice1NProbe; // This template will be used as a
                                           // probe in 1-N search
JaniceTemplate tmpl = NULL; // Best practice to initialize to NULL

if (janice_create_template(detections, num_detections, rolw, &tmpl) != JANICE_SUCCESS)
    // ERROR!
```

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(JaniceConstTemplate tmpl,
                                                        const char* attribute,
                                                        char** value);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
tmpl	JaniceConstTemplate	A template object to query the attribute from.
attribute	const char*	The name of the attribute to query.

value	char**	An uninitialized pointer to hold the attribute value. The implementor should allocate this object during the function call. The returned value must be a null terminated string.

janice_serialize_template

Serialize a JaniceTemplate object to a flat buffer.

Signature

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

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
tmpl	JaniceConstTemplate	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 JaniceBuffer	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(JaniceConstTemplate tmpl,
                                                const char* filename);
```

ThreadSafety

This function is reentrant.

Parameters

Name	Type	Description
tmpl	JaniceConstTemplate	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. Best practice dictates the pointer should be set to *NULL* after it is free.

Example

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

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) 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;
```

JaniceConstGallery

A pointer to a constant JaniceGalleryType object.

Signature

```
typedef const struct JaniceGalleryType* JaniceConstGallery;
```

JaniceTemplateId

A unique identifier for a template.

Signature

```
typedef uint32_t JaniceTemplateId;
```

JaniceTemplateIds

An array of JaniceTemplateId objects.

Signature

```
typedef JaniceTemplateId* JaniceTemplateIds;
```

Functions

janice_create_gallery

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

Signature

```
JANICE_EXPORT JaniceError janice_create_gallery(JaniceConstTemplates tmpls,
                                                const JaniceTemplateIds ids,
                                                uint32_t num_tmpls,
                                                JaniceGallery* gallery);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
tmpls	JaniceConstTemplates	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, leaving the templates in a valid state after this operation.
ids	const JaniceTemplateIds	A set of unique indentifiers to associate with the input templates. The *ith* id in this array corresponds to the *ith* input template. This array must be the same length as *tmpls*. If *tmpls* is *NULL* this object should also be *NULL*
num_tmpls	uint32_t	The length of *tmpls* and *ids*.
gallery	JaniceGallery *	An uninitialized gallery object. The implementor should allocate this object during the function call. The user is required to free the 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.

Signature

```
JANICE_EXPORT JaniceError janice_gallery_reserve(size_t n);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
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,
                                                JaniceConstTemplate tmpl,
                                                JaniceTemplateId id);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
gallery	JaniceGallery	A gallery object to insert the template into.
tmpl	JaniceConstTemplate	A template object to insert into the gallery. The template has the role Janice1NGallery. The template should be copied into the gallery. It must remain in a valid state after this function call.
id	JaniceTemplateId	A unique id to associate with the input template.

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_remove

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

Signature

```
JANICE_EXPORT JaniceError janice_gallery_remove(JaniceGallery gallery,
                                                uint32_t id);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
gallery	JaniceGallery	A gallery object to remove a template from. The template to remove is indicated by its unique id.
id	JaniceTemplateId	A unique id associated with a template int the gallery.

Example

```
JaniceTemplate tmpl; // Where tmpl is a valid template object created
                     // previously
const 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_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	JaniceConstGallery	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	JaniceConstGallery	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. Best practice dicates the pointer should be set to *NULL* after it is freed.

Example

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

Comparison

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.

Typedefs

JaniceSimilarity

A number representing the similarity between two templates. See [functions.md#JaniceVerifySimilarityScore] for more information.

Signature

```
typedef double JaniceSimilarity
```

JaniceSimilarities

An array of JaniceSimilarity objects.

Signature

```
typedef JaniceSimilarity* JaniceSimilarities;
```

JaniceSearchTemplateIds

An array of JaniceTemplateId objects.

Signature

```
typedef JaniceTemplateId* JaniceSearchTemplateIds;
```

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	JaniceConstTemplate	A reference template. This template was created with the *Janice11Reference* role.
verification	JaniceConstTemplate	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_search

Compute 1-N search results between a query template object and a target gallery object. The function allocates two arrays of equal size, one containing Similarity Score and the other containing the unique id of the template the score was computed with (along with the query). Often it is desirable (and perhaps computationally efficient) to only see the top K scores out of N possible templates. The option to set a K is provided to the user as part of the function parameters.

Signature

```
JANICE_EXPORT JaniceError janice_search(JaniceConstTemplate probe,
                                       JaniceConstGallery gallery,
                                       uint32_t num_requested,
                                       JaniceSimilarities* similarities,
                                       JaniceSearchTemplateIds* ids,
                                       uint32_t* num_returned);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
probe	JaniceConstTemplate	A template to use as a query. The template was created with the Janice1NProbe role.
gallery	JaniceConstGallery	A gallery object to search against.
num_requested	uint32_t	The number of requested number of returns. If the user would like as many returns as there are templates in the gallery they can set this to 0.
similarities	:ref`JaniceSimilarities` *	An uninitialized array of similarity scores. The scores must be in descending order (i.e. the highest score is stored at index 0). The implementor should allocate this object during the function call. The user is responsible for freeing the object with janice_free_similarities.
ids	JaniceSearchTemplateIds *	An uninitialized array of unique ids identifying the target templates associated with each score in *similarities*. This array must be the same size as *similarities*. The *ith* id in this array corresponds with the *ith* similarity in *similarities*. The implementor should allocate this object during the function call. The user is responsible for freeing the object by calling janice_free_search_ids.
num_returned	uint32_t*	The number of elements in the *similarities* and *ids* arrays. This number can be different from *num_requested*.

Example

```
JaniceTemplate probe;  // Where probe is a valid template object created
                       // previously
JaniceGallery gallery; // Where gallery is a valid gallery object created
                       // previously
const uint32_t num_requested = 50; // Request the top 50 matches
```

```
JaniceSimilarities similarities = NULL;
JaniceSearchTemplateIds ids = NULL;
uint32_t num_returned;

// Run search
if (janice_search(probe, gallery, num_requested, &similarities, &ids, &num_returned) != JANI
    // ERROR!

num_requested == num_returned; // This might not be true!
```

janice_free_similarities

Free any memory associated with a JaniceSimilarities object.

Signature

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

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
similarities	JaniceSimilarities *	An array of similarities to free.

janice_free_search_ids

Free any memory associated with a JaniceSearchTemplateIds object.

Signature

```
JANICE_EXPORT JaniceError janice_free_search_ids(JaniceSearchTemplateIds* ids);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
ids	JaniceSearchTemplateIds *	An array of ids to free.

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

JaniceMediaClusterItem

A structure that connects an input media and location with an output cluster. Because multiple detections could occur in a single piece of media a JaniceRect and a frame index are included to identify which detection this item refers to.

Fields

Name	Type	Description
cluster_id	JaniceClusterId	A unique identifier for the cluster this item belongs to. Items belonging to the same cluster should have the the id.
media_id	JaniceMediaId	A unique identifier for the media object this item corresponds to.
confidence	double	The confidence that this item belongs to this cluster.
rect	JaniceRect	The location of the clustered object in a frame or image.
frame	uint32_t	The frame index of the clustered object if the media is a video, otherwise 0.

JaniceTemplateClusterItem

A structure that connects an input template with an output cluster.

Fields

Name | Type		Description
cluster_id	JaniceClusterId	A unique identifier for the cluster this item belongs to. Items belonging to the same cluster should have the same id.
tmpl_id	JaniceTemplateId	A unique identifier for the template object this item corresponds to.
confidence	double	The confidence that this item belongs to this cluster.

Typedefs

JaniceClusterId

A unique identifier for a cluster

Signature

```
typedef uint32_t JaniceClusterId;
```

JaniceMediaId

A unique identifier for a media object

Signature

```
typedef uint32_t JaniceMediaId;
```

JaniceMediaIds

An array of JaniceMediaId objects.

Signature

```
typedef JaniceMediaId* JaniceMediaIds;
```

JaniceMediaClusterItems

An array of JaniceMediaClusterItem objects.

Signature

```
typedef struct JaniceMediaClusterItem* JaniceMediaClusterItems;
```

JaniceTemplateClusterItems

An array of JaniceTemplateClusterItem objects.

Signature

```
typedef struct JaniceTemplateClusterItem* JaniceTemplateClusterItems;
```

Function

janice_cluster_media

Cluster a collection of media objects into groups. Each media object may contain 0 or more objects of interest. To distinguish between these objects the output cluster contains an object location.

Signature

```
JANICE_EXPORT JaniceError janice_cluster_media(JaniceConstMedias input,
                                               const JaniceMediaIds input_ids,
                                               uint32_t num_inputs,
                                               uint32_t hint,
                                               JaniceMediaClusterItems* clusters,
                                               uint32_t* num_clusters);
```

Thread Safety

This function is reentrant.

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.

Parameters

Name	Type	Description
input	JaniceMediaIterators	An array of media objects to cluster.
input_ids	const JaniceMediaIds	An array of unique indentifiers for the input objects. This array must be the same size as *input*. The *ith* id should correspond to the *ith* media object in *input*.

num_inputs	uint32_t	The size of the *input* and *input_ids* arrays.
hint	uint32_t	A Hint to the clustering algorithm.
clusters	JaniceMediaClusterItems *	An uninitialized array of cluster items to store the result of clustering. The implementor should allocate this object during the function call. The user is responsible for freeing the object by calling janice_free_media_cluster_items.
num_clusters	uint32_t*	The size of the *clusters* array.

janice_cluster_templates

Cluster a collection of template objects into groups.

Signature

```
JANICE_EXPORT JaniceError janice_cluster_templates(JaniceConstTemplates input,
                                                   const JaniceTemplateIds input_ids,
                                                   uint32_t num_inputs,
                                                   uint32_t hint,
                                                   JaniceTemplateClusterItems* clusters,
                                                   uint32_t* num_clusters);
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
input	JaniceConstTemplates	An array of template objects to cluster.
input_ids	const JaniceTemplateIds	An array of unique indentifiers for the input objects. This array must be the same size as *input*. The *ith* id should correspond to the *ith* media object in *input*.
num_inputs	uint32_t	The size of the *input* and *input_ids* arrays.
hint	uint32_t	A Hint to the algorithm. The implementor may ignore this value if they choose.
clusters	JaniceTemplateClusterItems *	An uninitialized array to hold the cluster output. The implementor should allocate this object during the function call. The user is for freeing the object by calling janice_free_template_cluster_items.
num_clusters	uint32_t*	The size of the *clusters* array.

janice_free_media_cluster_items

Free any memory associated with a JaniceMediaClusterItems object.

Signature

```
JANICE_EXPORT JaniceError janice_free_media_cluster_items(JaniceMediaClusterItems* clusters)
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
clusters	JaniceMediaClusterItems *	The media cluster object to free.

janice_free_template_cluster_items

Free any memory associated with a JaniceTemplateClusterItems object.

Signature

```
JANICE_EXPORT JaniceError janice_free_template_cluster_items(JaniceTemplateClusterItems* clu
```

Thread Safety

This function is reentrant.

Parameters

Name	Type	Description
clusters	JaniceTemplateClusterItems *	The template cluster object to free.

License


```
/*****************************************************************************
 * Copyright (c) 2013 Noblis, Inc.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and/or associated documentation files (the
 * "Materials"), to deal in the Materials without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Materials, and to
 * permit persons to whom the Materials are furnished to do so, subject to
 * the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Materials.
 *
 * THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.
 *****************************************************************************/
```


Indices and tables

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