The OpenVX™ Export and Import Extension

The Khronos OpenVX Working Group, Editors: Steve Ramm, Radhakrishna Giduthuri

Version 1.1.1, Wed, 15 Aug 2018 06:03:12 +0000

Table of Contents

Chapter 1. Export and Import Extension to OpenVX 1.1

1.1. Purpose

Provide a way of exporting and importing pre-verified graphs or other objects, in vendor-specific formats.

1.2. Use Case

- Embedded systems using fixed graphs, to minimise the amount of code required.
- Safety-critical systems where the OpenVX library does not have a node API.
- CNN extensions which require the ability to import binary objects.

1.3. Acknowledgements

This specification would not be possible without the contributions from this partial list of the following individuals from the Khronos Working Group and the companies that they represented at the time:

- Radhakrishna Giduthuri - AMD
- Frank Brill - Cadence Design Systems
- Thierry Lepley - Cadence Design Systems
- Steve Ramm - Imagination Technologies
- Tomer Schwartz - Intel
- Jesse Villareal - Texas Instruments

Chapter 2. Requirements

2.1. Import and Export of Objects

- **2.1.1** Application must be able to specify which objects are to be exported.
- **2.1.2** The framework will also export any other objects required; for example, if a graph is to be exported then all components of that graph will also be exported even if their references were not given.
- **2.1.3** Upon Import, only those objects that were specified for export will be visible in the imported entity; all other objects may be present but are not directly accessible. For example, a pyramid object may be exported, in which case the levels of the pyramid will be available in the usual way. As another example, an image that is part of a pyramid is exported. Since an image has no API that allows accessing the pyramid of which it is part, then there is no requirement to export the rest of the pyramid.

2.2. Creation of Objects Upon Import

- **2.2.1** To make it possible to implement certain scenarios, for example when an image needs to be created in a different way in the import environment than the export environment, certain objects may be created by the application and passed to the import routine. These objects need to be specified at the time of export, and provided again at the time of import.
- **2.2.2** All other required objects will be created by the framework upon import.

2.3. Import and Export of Data Values For Objects To Be Created By The Framework

- **2.3.1** Some objects may contain data values (as distinct from Meta Data) that require preservation across the export and import routines. The application can specify this at the time of export; those objects which are listed (by giving references) for export will then either be stripped of data values or have their data values entirely exported.
- **2.3.2** For those objects which are **not** listed, the following rules apply:
  - **2.3.2.1** In any one graph, if an object is not connected as an output parameter then its data values will be exported (and imported).
  - **2.3.2.2** Where the object in (1) is a composite object such as a Pyramid or ObjectArray, then rule (1) applies to all sub-objects by definition.
  - **2.3.2.3** Where the object in (1) is a sub-object such as a Region Of Interest or member of an ObjectArray, and the composite object does not meet the conditions of rule (1), then rule (1) applies to the sub-object only.
  - **2.3.2.4** When objects are imported, the exported data values are assigned. However, if parts of the data were not defined at the time of export, then there is no guarantee that upon import the same values will be present. For example, consider an image where values had been written only to some (rectangular) part of the image before export. After import, only

this part of the image will be guaranteed to contain the same values; those parts which were undefined before will be set to the default value for the data field. In the absence of any other definition the default value is zero.

- **2.3.3** For those objects which **are** listed, then:
  - **2.3.3.1** If the application requires, all defined values shall be exported.
  - **2.3.3.2** The application requires, no values need be exported. The behavior here is as though no values had been defined (written) for the object.
  - **2.3.3.3** Areas of undefined values will remain undefined (and possibly containing different random values) upon import.
  - **2.3.3.4** All values are initialized upon import. If data was not defined, then it is set to the default value for the data field. In the absence of any other definition the default value is zero.

2.4. Import and Export of Values For Objects To Be Created By The Application

- **2.4.1** Objects created by the application before import of the binary object must have their data values defined by the application before the import operation.
- **2.4.2** Sometimes changing the value stored in an object that is an input parameter of a verified graph will require that the graph is verified again before execution. If such an object is listed as to be supplied by the application, then the export operation will fail.

2.5. Import and Export of Meta Data

- **2.5.1** For all objects that are visible in the import, all query-able Meta Data must appear the same after import as before export.
- **2.5.2** Objects created by the application before import and provided to the import API must match in type, size, etc. and therefore the export must export sufficient information for this check to be done.
- **2.5.3** An Import may fail if the application-provided objects do not match those given at the time of export.
- **2.5.4** Graphs with delays that are registered for auto-ageing at the time of export will be in the same condition after import of the objects.

2.6. Restrictions Upon What References May Be Exported

- **2.6.1** Export will fail if a `vx_context` is given in a list to export.
- **2.6.2** Export will fail if a `vx_import` is given in a list to export. (`vx_import` is the type of the object returned by the import functions).
- **2.6.3** Export will fail if a reference to a virtual object is given in the list to export.

- **2.6.4** Export will fail if a `vx_node`, `vx_kernel`, or `vx_parameter` is given in the list to export.
- **2.6.5** Export is otherwise defined for “objects” in the OpenVX 1.1 specification.

2.7. Access To Object References In The Imported Object

- **2.7.1** References are obtained from the import API for those objects whose references were listed at the time of export. These are not the same objects; they are equivalent objects created by the framework at import time. The implementation guarantees that references will be available and valid for all objects listed at the time of export, or the import will fail.
- **2.7.2** References additionally may be obtained using a name given to an object before export.
- **2.7.3** Before export, objects may be named for retrieval by name using the existing API

```
vx_status vxSetReferenceName(
    vx_reference                                ref,
    const vx_char*                              name);
```

- **2.7.4** Export will fail if duplicate names are found for listed references.
- **2.7.5** Import will fail if duplicate names are found in the import object.
- **2.7.6** If references are obtained by name, only those objects whose references were listed at the time of export can be found by name.
- **2.7.7** A `vx_node`, `vx_kernel`, or `vx_parameter` cannot be obtained from the import object.

2.8. "Deployment" Feature Sub-set

The deployment feature subset consists of the following APIs: Import Objects From Memory API .

Chapter 3. Module Documentation

3.1. Export To Memory API

Export objects to a location in memory.

Macros

- VX_IX_USE_APPLICATION_CREATE
- VX_IX_USE_EXPORT_VALUES
- VX_IX_USE_NO_EXPORT_VALUES

Functions

- vxExportObjectsToMemory
- vxReleaseExportedMemory

3.1.1. Macros

VX_IX_USE_APPLICATION_CREATE

How to export and import an object.

```
#define VX_IX_USE_APPLICATION_CREATE      (VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IX_USE)
+ 0x0)
```

The application will create the object before import.

VX_IX_USE_EXPORT_VALUES

How to export and import an object.

```
#define VX_IX_USE_EXPORT_VALUES           (VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IX_USE)
+ 0x1)
```

Data values are exported and restored upon import.

VX_IX_USE_NO_EXPORT_VALUES

How to export and import an object.

```
#define VX_IX_USE_NO_EXPORT_VALUES        (VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IX_USE)
+ 0x2)
```

Data values are not exported.

3.1.2. Functions

vxExportObjectsToMemory

Exports selected objects to memory in a vendor-specific format.

```
vx_status vxExportObjectsToMemory(
    vx_context                                  context,
    vx_size                                     numrefs,
    const vx_reference*                         refs,
    const vx_enum*                              uses,
    const vx_uint8**                            ptr,
    vx_size*                                    length);
```

A list of references in the given context is supplied to this function, and all information required to re-create these is stored in memory in such a way that those objects may be re-created with the corresponding import function, according to the usage specified by the *uses* parameter.

The information must be context independent in that it may be written to external storage for later retreival with another instantiation of a compatible implementation.

The list of objects to export may contain only valid references (i.e. vxGetStatus() will return VX_SUCCESS) to `vx_graph` and non-virtual data objects or the function will fail. (Specifically not `vx_context`, `vx_import`, `vx_node`, `vx_kernel`, `vx_parameter` or `vx_meta_format`)

Some node creation functions take C parameters rather than OpenVX data objects (such as the *gradient_size* parameter of `vxHarrisCornersNode` that is provided as a `vx_int32`), because these are intended to be fixed at node creation time; nevertheless OpenVX data objects may be assigned to them, for example if the `vxCreateGenericNode` API is used. A data object corresponding to a node parameter that is intended to be fixed at node creation time must not be in the list of exported objects nor attached as a graph parameter or the export operation will fail.

The *uses* array specifies how the objects in the corresponding *refs* array will be exported. A data object will always have its meta-data (e.g. dimensions and format of an image) exported, and optionally may have its data (e.g. pixel values) exported, and additionally you can decide whether the importing application will create data objects to replace those attached to graphs, or if the implementation will automatically create them.

Enumeration for *uses*	**Definition**
`VX_IX_USE_APPLICATION_CREATE`	Export sufficient data to check that an application-supplied object is compatible when the data is later imported.
`VX_IX_USE_EXPORT_VALUES`	Export complete information (for example image data or value of a scalar).
`VX_IX_USE_NO_EXPORT_VALUES`	Export meta-data only; the importing application will set values as applicable

Note

The `VX_IX_USE_APPLICATION_CREATE` value must be given for images created from handles, or the the export operation will fail.

The values in *uses* are applicable only for data objects and are ignored for `vx_graph` objects.

If the list *refs* contains `vx_graph` objects, these graphs will be verified during the export operation and the export operation will fail if verification fails; when successfully exported graphs are subsequently imported they will appear as verified.

Note

The implementation may also choose to re-verify any previously verified graphs and apply optimisations based upon which references are to be exported and how.

Any data objects attached to a graph that are hidden, i.e. their references are not in the list *refs*, may be treated by the implementation as virtual objects, since they can never be visible when the graph is subsequently imported.

Note that imported graphs cannot become unverified. Attempts to change the graph that might normally cause the graph to be unverified, e.g. calling vxSetGraphParameterByIndex with an object with different metadata, will fail.

The implementation should make sure that all permissible changes of exported objects are possible without re-verification. For example:

- A uniform image may be swapped for a non-uniform image, so corresponding optimisations should be inhibited if a uniform image appears in the *refs* list
- An image that is a region of interest of another image may be similarly replaced by any other image of matching size and format, and vice-versa

If a graph is exported that has delays registered for auto-aging, then this information is also exported.

If the function is called with NULL for any of its parameters, this is an error.

The reference counts of objects as visible to the calling application will not be affected by calling this function.

The export operation will fail if more than one object whose reference is listed at *refs* has been given the same non-zero length name (via `vxSetReferenceName`).

If a graph listed for export has any graph parameters not listed at *refs*, then the export operation will fail.

Note

The order of the references supplied in the *refs* array will be the order in which the framwork will supply references for the corresponding import operation with `vxImportObjectsFromMemory`.

The same length of *uses* array, containing the same values, and the same value of *numrefs*, must be supplied for the corresponding import operation.

For objects not listed in *refs*, the following rules apply:

1. In any one graph, if an object is not connected as an output of a node in a graph being exported then its data values will be exported (for subsequent import).
2. Where the object in (1) is a composite object (such as a pyramid) then rule (1) applies to all of its sub-objects.
3. Where the object in (1) is a sub-object such as a region of interest, and the composite object (in this case the parent image) does not meet the conditions of rule (1), then rule (1) applies to the sub-object only.

Parameters

- `[in]` *context* - context from which to export objects, must be valid .
- `[in]` *numrefs* - number of references to export.
- `[in]` *refs* - references to export. This is an array of length numrefs populated with the references to export.
- `[in]` *uses* - how to export the references. This is an array of length numrefs containing values as described above.
- `[out]` *ptr* - returns pointer to binary buffer. On error this is set to NULL.
- `[out]` *length* - number of bytes at *ptr. On error this is set to zero.

Returns: A `vx_status` value.

Return Values

- `VX_SUCCESS` - If no errors occurred and the objects were sucessfully exported. An error is indicated when the return value is not VX_SUCCESS.

An implementation may provide several different return values to give useful diagnostic information in the event of failure to export, but these are not required to indicate possible recovery mechanisms, and for safety critical use assume errors are not recoverable.

Postcondition: `vxReleaseExportedMemory` is used to deallocate the memory.

vxReleaseExportedMemory

Releases memory allocated for a binary export when it is no longer required.

```
vx_status vxReleaseExportedMemory(
    vx_context                                  context,
    const vx_uint8**                            ptr);
```

This function releases memory allocated by vxExportObjectsToMemory.

Parameters

- [in] *context* - The context for which vxExportObjectsToMemory was called.
- [inout] *ptr* - A pointer previously set by calling vxExportObjectsToMemory. The function will fail if *ptr does not contain an address of memory previously allocated by vxExportObjectsToMemory.

Postcondition: After returning from sucessfully from this function *ptr is set to NULL.

Returns: A vx_status value.

Return Values

- VX_SUCCESS - If no errors occurred and the memory was sucessfully released.

An error is indicated when the return value is not VX_SUCCESS.

An implementation may provide several different return values to give useful diagnostic information in the event of failure to export, but these are not required to indicate possible recovery mechanisms, and for safety critical use assume errors are not recoverable.

Precondition: vxExportObjectsToMemory is used to allocate the memory.

3.2. Import Objects From Memory API

Import objects from memory.

Macros

- VX_TYPE_IMPORT

Typedefs

- vx_import

Functions

- vxGetImportReferenceByName
- vxImportObjectsFromMemory
- vxReleaseImport

3.2.1. Macros

VX_TYPE_IMPORT

The Object Type Enumeration for import.

```
#define VX_TYPE_IMPORT                  0x814
```

A vx_import.

3.2.2. Typedefs

vx_import

The Import Object. Import is a container of OpenVX objects, which may be retreived by name.

```
typedef struct _vx_import *vx_import;
```

3.2.3. Functions

vxGetImportReferenceByName

Get a reference from the import object by name.

```
vx_reference vxGetImportReferenceByName(
    vx_import                                   import,
    const vx_char*                              name);
```

All accessible references of the import object created using vxImportObjectsFromMemory are in the array *refs*, which is populated partly by the application before import, and partly by the framework. However, it may be more convenient to access the references in the import object without referring to this array, for example if the import object is passed as a parameter to another function. In this case, references may be retreived by name, assuming that vxSetReferenceName was called to assign a name to the reference. This function searches the given import for the given name and returns the associated reference.

The reference may have been named either before export or after import.

If more than one reference exists in the import with the given name, this is an error.

Only references in the array *refs* after calling vxImportObjectsFromMemory may be retrieved using this function.

A reference to a named object may be obtained from a valid import object using this API even if all other references to the object have been released.

Parameters

- [in] *import* - The import object in which to find the name; the function will fail if this parameter

is not valid.

- `[in]` *name* - The name to find, points to a string of at least one and less than VX_MAX_REFERENCE_NAME bytes followed by a zero byte; the function will fail if this is not valid.

Returns: A `vx_reference`.

Calling `vxGetStatus` with the reference as a parameter will return VX_SUCCESS if the function was successful. : Another value is given to indicate that there was an error.

On success, the reference count of the object in question is incremented.

An implementation may provide several different error codes to give useful diagnostic information in the event of failure to retrieve a reference, but these are not required to indicate possibly recovery mechanisms, and for safety critical use assume errors are not recoverable.

Precondition: `vxSetReferenceName` was used to name the reference.

Postcondition: use `ref vxReleaseReference` or appropriate specific release function to release a reference obtained by this method.

vxImportObjectsFromMemory

Imports objects into a context from a vendor-specific format in memory.

```
vx_import vxImportObjectsFromMemory(
    vx_context                                  context,
    vx_size                                     numrefs,
    vx_reference*                               refs,
    const vx_enum*                              uses,
    const vx_uint8*                             ptr,
    vx_size                                     length);
```

This function imports objects from a memory blob previously created using `vxExportObjectsToMemory`.

A pointer to memory is given where a list of references is stored, together with the list of uses which describes how the references are used. The number of references given and the list of uses must match that given upon export, or this function will not be sucessful.

The *uses* array specifies how the objects in the corresponding *refs* array will be imported:

`VX_IX_USE_APPLICATION_CREATE`	The application must create the object and supply the reference; the meta-data of the object must match exactly the meta-data of the object when it was exported, except that the name need not match. If the supplied reference has a different name to that stored, the supplied name is used.

VX_IX_USE_EXPORT_VALUES	The implementation will create the object and set the data in it. Any data not defined at the time of export of the object will be set to a default value (zero in the absence of any other definition) upon import.
VX_IX_USE_NO_EXPORT_VALUES	The implementation will create the object and the importing application will set values as applicable.

References are obtained from the import API for those objects whose references were listed at the time of export. These are not the same objects; they are equivalent objects created by the framework at import time. The implementation guarantees that references will be available and valid for all objects listed at the time of export, or the import will fail.

The import operation will fail if more than one object whose reference is listed at *refs* has been given the same non-zero length name (via vxSetReferenceName).

The import will be unsuccessful if any of the parameters supplied is NULL.

After completion of the function the memory at *ptr* may be deallocated by the application as it will not be used by any of the created objects.

Any delays imported with graphs for which they are registered for auto-aging remain registered for auto-aging.

After import, a graph must execute with exactly the same effect with respect to its visible parameters as before export.

Note

The *refs* array must be the correct length to hold all references of the import; this will be the same length that was supplied at the time of export. Only references for objects created by the application, where the corresponding *uses* entry is VX_IX_USE_APPLICATION_CREATE should be filled in by the application; all other entries will be supplied by the framework and may be initialised by the application to NULL. The *uses* array must have the identical length and content as given at the time of export, and the value of *numrefs* must also match; these measures increase confidence that the import contains the correct data.

Note

Graph parameters may be changed after import by using the `vxSetGraphParameterByIndex` API, and images may also be changed by using the `vxSwapImageHandle` API. When `vxSetGraphParameterByIndex` is used, the framework will check that the new parameter is of the correct type to run with the graph, which cannot be re-verified. If the reference supplied is not suitable, an error will be returned, but there may be circumstances where changing graph parameters for unsuitable ones is not detected and could lead to implementation-dependent behaviour; one such circumstance is when the new parameters are images corresponding to overlapping regions of interest. The user should avoid these circumstances. In other words,

- The meta data of the new graph parameter must match the meta data of the graph parameter it replaces.
- A graph parameter must not be NULL.

Parameters

- `[in]` *context* - context into which to import objects, must be valid
- `[in]` *numrefs* - number of references to import, must match export
- `[inout]` *refs* - references imported or application-created data which must match meta-data of the export
- `[in]` *uses* - how to import the references, must match export values
- `[in]` *ptr* - pointer to binary buffer containing a valid binary export
- `[in]` *length* - number of bytes at *ptr, i.e. the length of the export

Returns: A `vx_import`. Calling `vxGetStatus` with the `vx_import` as a parameter will return VX_SUCCESS if the function was successful.

Another value is given to indicate that there was an error.

An implementation may provide several different error codes to give useful diagnostic information in the event of failure to import objects, but these are not required to indicate possibly recovery mechanisms, and for safety critical use assume errors are not recoverable.

Postcondition: `vxReleaseImport` is used to release the import object.

Postcondition: Use `vxReleaseReference` or an appropriate specific release function to release the references in the array refs when they are no longer required.

vxReleaseImport

Releases an import object when no longer required.

```
vx_status vxReleaseImport(
    vx_import*                                  import);
```

This function releases the reference to the import object.

Other objects including those imported at the time of creation of the import object are unaffected.

Parameters

- `[inout]` *import* - The pointer to the reference to the import object.

Postcondition: After returning sucessfully from this function the reference is zeroed.

Returns: A `vx_status` value.

Return Values

- `VX_SUCCESS` - If no errors occurred and the import was sucessfully released.

An error is indicated when the return value is not VX_SUCCESS.

An implementation may provide several different return values to give useful diagnostic information in the event of failure to export, but these are not required to indicate possibly recovery mechanisms, and for safety critical use assume errors are not recoverable.

Precondition: `vxImportObjectsFromMemory` is used to create an import object.