# The **OpenVX™** Export and Import Extension

Version 1.0 (Provisional)

Document Revision: 2b213f9

Khronos Vision Working Group

*Editor:* Steve Ramm
*Editor:* Xin Wang

Copyright ©2016 The Khronos Group Inc.

Copyright ©2016 The Khronos Group Inc. All Rights Reserved.

This specification is protected by copyright laws and contains material proprietary to the Khronos Group, Inc. It or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast, or otherwise exploited in any manner without the express prior written permission of the Khronos Group. You may use this specification for implementing the functionality therein, without altering or removing any trademark, copyright or other notice from the specification, but the receipt or possession of this specification does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part. Khronos Group grants express permission to any current Promoter, Contributor or Adopter member of Khronos to copy and redistribute UNMODIFIED versions of this specification in any fashion, provided that NO CHARGE is made for the specification and the latest available update of the specification for any version of the API is used whenever possible. Such distributed specification may be reformatted AS LONG AS the contents of the specification are not changed in any way. The specification may be incorporated into a product that is sold as long as such product includes significant independent work developed by the seller. A link to the current version of this specification on the Khronos Group website should be included whenever possible with specification distributions.

Khronos Group makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this specification, including, without limitation, any implied warranties of merchantability or fitness for a particular purpose or non-infringement of any intellectual property. Khronos Group makes no, and expressly disclaims any, warranties, express or implied, regarding the correctness, accuracy, completeness, timeliness, and reliability of the specification. Under no circumstances will the Khronos Group, or any of its Promoters, Contributors or Members or their respective partners, officers, directors, employees, agents or representatives be liable for any damages, whether direct, indirect, special or consequential damages for lost revenues, lost profits, or otherwise, arising from or in connection with these materials. SAMPLE CODE and EXAMPLES, as identified herein, are expressly depicted herein with a ”grey” watermark and are included for illustrative purposes only and are expressly outside of the Scope as defined in Attachment A - Khronos Group Intellectual Property (IP) Rights Policy of the Khronos Group Membership Agreement. A Member or Promoter Member shall have no obligation to grant any licenses under any Necessary Patent Claims covering SAMPLE CODE and EXAMPLES.

# 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 Variations of This Extension

Some of the following requirements apply separately to Safety-Critical and Non-Safety-Critical implementations. These are annotated [SC] and [Non-SC] respectively.

A separate section lists those requirements that may be treated as a subset of the extension for the purposes of providing a smaller, more easily verified “deployment” implementation.

Implementations may optionally provide variations of this extension which allow import and export in defined formats, for portability purposes.

# 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 Object-Array, 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** [Non-SC] 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 still be undefined and may contain different random values.

  - **2.3.2.5** [SC] 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.

- **2.3.3** 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** [SC] 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** [SC] 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.
- **2.4.3** [Non-SC] 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 a graph exists in an imported object and the value of the input parameter has been changed, then any attempts to execute the graph may result in a vendor-specific outcome.

## 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** [SC] Export will fail if a reference to a virtual object is given in the list to export.
- **2.6.4** [SC] 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** [Non-SC] 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 application should use the vxGetStatus() API to check the validity of references before use; the framework may give errors such as VX_ERROR_OPTIMIZED_AWAY if appropriate. Application designers should take care to ensure that these sorts of situations are tested before deployment.
- **2.7.2** [SC] 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.
- **2.7.3** References additionally may be obtained using a name given to an object before export.
- **2.7.4** Before export, objects may be named for retrieval by name using the existing API *vx_status vxSetReferenceName(vx_reference ref, const vx_char * name).*
- **2.7.5** [SC] Export will fail if duplicate names are found for listed references.
- **2.7.6** [SC] Import will fail if duplicate names are found in the import object.
- **2.7.7** [SC] If references are obtained by name, only those objects whose references were listed at the time of export can be found by name.
- **2.7.8** [SC] 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 API – From Memory, Release Import API, Get Import Reference API and their dependent requirements, and optionally Export Objects To URL and Import Objects From URL.

## 2.9 Future Directions

Sections Export And Import From URL, Export Objects To URL, and Import Objects From URL describe a mechanism whereby the source or destination of the data is under application control. In no part of this extension is the format of the data described; that is left to the implementation.

However, sections Export Objects To URL and Import Objects From URL do mention that the implementation may examine the url to determine what format to use. Currently we make no specification here. Possible future versions of this specification may make a requirement that if the url is formatted in a certain way, then the output or input should conform to a specific format. For this reason, queries starting with the string “khr” are reserved for future use. One possible application would be to allow export and import in a portable format such as xml.

In section Restrictions Upon What References May Be Exported we disallow the export of a vx_context. This restriction may be removed in future, but there will be many other associated changes, and it may be applicable only to certain clearly defined formats.

# Chapter 3

# Module Documentation

## 3.1 Reference Export / Import Type

### Typedefs

- typedef struct _vx_import * vx_import

### Enumerations

- enum vx_ext_import_type_e { VX_TYPE_IMPORT = 0x815 }
- enum _vx_export_use_e {
  VX_EXPORT_USE_APPLICATION_CREATE,
  VX_EXPORT_USE_EXPORT_VALUES,
  VX_EXPORT_USE_NO_EXPORT_VALUES }

### 3.1.1 Detailed Description

Associated with each reference supplied to the export and import API is a description of how the reference is used. A new enumeration defines this.

### 3.1.2 Enumeration Type Documentation

**enum _vx_export_use_e**

Enumerator

***VX_EXPORT_USE_APPLICATION_CREATE*** The application will create the object before import.

***VX_EXPORT_USE_EXPORT_VALUES*** Data values to be exported and restored on import.

***VX_EXPORT_USE_NO_EXPORT_VALUES*** Data values will not be exported.

Definition at line 54 of file vx_import_export.h.

**enum vx_ext_import_type_e**

The Object Type Enumeration for import.

Enumerator

***VX_TYPE_IMPORT*** A `vx_import`

Definition at line 46 of file vx_import_export.h.

## 3.2 Export API – To Memory

### Functions

- VX_API_ENTRY vx_status VX_API_CALL vxExportObjectsToMemory (vx_context context, vx_size numrefs, const vx_reference *refs, const vx_enum *uses, const vx_uint8 **ptr, vx_size *length)

### 3.2.1 Detailed Description

Export objects to a location in memory.

- **3.2.1.1** The function to export objects to a location in memory is as the function documentation section.
- **3.2.1.2** If the list of references contains one or more `vx_graph` , then these are first verified if necessary. (Any optimizations made by the framework should take into account which references are to be exported and how).
- **3.2.1.3** If the function is called with NULL for any of the parameters, it will return an error.
- **3.2.1.4** `vxExportObjectsToMemory` will create a dump of the objects at a memory location, which it returns in the ptr parameter, taking account of the uses specified with the uses parameter.
- **3.2.1.5** The size of the memory dump will be returned at the location specified by the length parameter.
- **3.2.1.6** If the export is successful (see also preceding sections), the return value is VX_SUCCESS.
- **3.2.1.7** [SC] A non-recoverable error is indicated when the return value is not VX_SUCCESS.
- **3.2.1.8** 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.
- **3.2.1.9** The reference counts of the objects listed for export will not be affected by calling this function.
- **3.2.1.10** [SC] It is a pre-requisite that all pointers passed to this function are valid. The function should attempt to validate that all pointers are valid and that the references in the given array are valid.
- **3.2.1.11** To be conformant to this extension specification an implementation must implement this function.

### 3.2.2 Function Documentation

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

**Parameters**

| | | |
|---|---|---|
| `in` | *context* | Context from which to export objects. |
| `in` | *numrefs* | Number of references to export. |
| `in` | *refs* | References to export. |
| `in` | *uses* | How to export the references, use vx_export_use_e. |
| `out` | *ptr* | Returns pointer to binary buffer. |
| `out` | *length* | Number of bytes at *ptr |

Returns

`vx_status` A vx_status_e enumeration.

## 3.3 Release Exported Memory API

### Functions

- VX_API_ENTRY vx_status VX_API_CALL vxReleaseExportedMemory (vx_context context, const vx_uint8 **ptr)

### 3.3.1 Detailed Description

Release the memory allocated by `vxExportObjectsToMemory`.

- **3.3.1.1** The function to release the memory allocated by `vxExportObjectsToMemory` is as the function documentation section.
- **3.3.1.2** This function will release the memory previously allocated for an export at **ptr.
- **3.3.1.3** The function will fail if *ptr does not contain the address of memory previously allocated by an export operation.
- **3.3.1.4** On success, *ptr will be set to NULL and the function will return VX_SUCCESS.
- **3.3.1.5** [SC] A non-recoverable error is indicated when the return value is not VX_SUCCESS.
- **3.3.1.6** 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.
- **3.3.1.7** To be conformant to this extension specification an implementation must implement this function.

### 3.3.2 Function Documentation

**VX_API_ENTRY vx_status VX_API_CALL vxReleaseExportedMemory ( vx_context *context,* const vx_uint8 ** ***ptr*** )**

Used to release exported memory.

**Parameters**

| | | |
|---|---|---|
| `in` | *context* | Context from which to export objects. |
| `in,out` | *ptr* | Pointer returned by vxExportObjectsToMemory. |

Returns

`vx_status` A vx_status_e enumeration.

## 3.4 Import Objects API – From Memory

### Functions

- VX_API_ENTRY vx_import VX_API_CALL vxImportObjectsFromMemory (vx_context context, vx_size numrefs, vx_reference *refs, const vx_enum *uses, const vx_uint8 *ptr, vx_size length)

### 3.4.1 Detailed Description

Import objects from memory.

- **3.4.1.1** The function to import objects from memory is as the function documentation section.
- **3.4.1.2** If the function is called with NULL for any of the parameters, it will return an error.
- **3.4.1.3** A non-recoverable error is indicated when the return value is NULL, otherwise the return value is a valid vx_import. (see also preceding sections for causes of failure)
- **3.4.1.4** The import will fail if the number of references given on import does not match the number of references given on export.
- **3.4.1.5** The import will fail if the list of uses does not match the list of uses given upon export.
- **3.4.1.6** On successful import, references will be written to those entries in the array refs where the corresponding entry in the array uses is not VX_EXPORT_USE_APPLICATION_CREATE.
- **3.4.1.7** The application must create any objects marked with a use of VX_EXPORT_USE_APPLICATION_CREATE and the import will fail if such objects do not sufficiently match those given on export.
- **3.4.1.8** After a call to vxImportObjectsFromMemory the memory at ptr may be re-used or deallocated by the application, it is not required by the import object.
- **3.4.1.9** Releasing the import object does not automatically release the references at refs – they are not "borrowed".
- **3.4.1.10** Releasing the references at refs does not mean that the objects are deallocated, since references may also exist in the import object.
- **3.4.1.11** To be conformant to this extension specification an implementation must implement this function.

### 3.4.2 Function Documentation

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

**Parameters**

| | | |
|---|---|---|
| in | *context* | Context into which to import objects. |
| in | *numrefs* | Number of references to import. |
| in,out | *refs* | References imported. |
| in | *uses* | To import the references, use vx_export_use_e. |
| in | *ptr* | Where to get the data. |
| in | *length* | Number of bytes at ptr. |

Returns

vx_import

**Return values**

| | |
|---|---|
| *0* | Invalid index. |
| $*$ | An import reference. |

## 3.5 Release Import API

### Functions

- VX_API_ENTRY vx_status VX_API_CALL vxReleaseImport (vx_import *import)

### 3.5.1 Detailed Description

This function is provided to release a reference to an import object; it also internally releases its references to its imported objects. These imported objects may not be garbage collected until their total reference counts are zero.

- **3.5.1.1** This function is provided to release a reference to an import object; it also internally releases its references to its imported objects. These imported objects may not be garbage collected until their total reference counts are zero. The function is as the function documentation section.
- **3.5.1.2** If the import reference is successfully released, the return value is VX_SUCCESS.
- **3.5.1.3** [SC] A non-recoverable error is indicated when the return value is not VX_SUCCESS.
- **3.5.1.4** 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.
- **3.5.1.5** To be conformant to this extension specification an implementation must implement this function.

### 3.5.2 Function Documentation

**VX_API_ENTRY vx_status VX_API_CALL vxReleaseImport ( vx_import * *import* )**

To release an import object after use.

Returns

`vx_status` A vx_status_e enumeration.

## 3.6 Get Import Reference API

### Functions

- VX_API_ENTRY vx_reference
  VX_API_CALL vxGetImportReferenceByName (vx_import import, const vx_char *name)

### 3.6.1 Detailed Description

Defines a companion API to retrieve references by name.

- **3.6.1.1** All the references required to use the import object are in the array refsrefs (as defined as a parameter to the import functions in sections 3.4 and 3.8), which is populated partially by the application before the 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 to be passed as a parameter to another function. For this, the object naming API mentioned in 2.7.4 is used.
- **3.6.1.2** A non-recoverable error is indicated when the return value is NULL, otherwise it is a valid reference.
- **3.6.1.3** Possible errors include invalid import reference, name not found, etc.
- **3.6.1.4** [Non-SC] If more than one reference exists with the given name, one of the references will be returned.
- **3.6.1.5** [SC] If more than one reference exists with the given name, this is an error and NULL is returned.
- **3.6.1.6** To be conformant to this extension specification an implementation must implement this function.

### 3.6.2 Function Documentation

**VX_API_ENTRY vx_reference VX_API_CALL vxGetImportReferenceByName ( vx_import *import,* const vx_char * *name* )**

To get a reference from the import object by name.

Returns

`vx_reference`.

## 3.7 Export And Import From URL

### Typedefs

- typedef struct _vx_url_handle * vx_url_handle
- typedef vx_url_handle(VX_CALLBACK * vx_url_create_f )(const vx_char *url)
- typedef vx_url_handle(VX_CALLBACK * vx_url_open_f )(const vx_char *url)
- typedef vx_status(VX_CALLBACK * vx_url_write_f )(vx_url_handle h, const vx_uint8 *ptr, vx_size size)
- typedef vx_status(VX_CALLBACK * vx_url_read_f )(vx_url_handle h, vx_uint8 *ptr, vx_size size, vx_size *bytes_read)
- typedef vx_status(VX_CALLBACK * vx_url_close_f )(vx_url_handle h)

### 3.7.1 Detailed Description

Definitions for URL I/O.

- **3.7.1.1** The typedefs given above are provided to assist in export of objects to and import from an arbitrary URL. This can be utilized by an application for example to write to a filing system or to read from a memory device on a serial interface. The application(s) should provide the call-back functions which will conform to the requirements of this section.
- **3.7.1.2** The url will be in the standard form as defined at $https://url.spec.whatwg.org/:scheme:[//[user:password@]host[:port]][/]path[?query][fragment]$.
- **3.7.1.3** If a query is used it should not start with the string “khr”.
- **3.7.1.4** `vx_url_create_f` will create a `vx_url_handle`which will be NULL in the case of a non-recoverable error, otherwise it will be a valid useable `vx_url_handle` for output.
- **3.7.1.5** `vx_url_open_f` will create a vx_url_handle which will be NULL in the case of a non-recoverable error, otherwise it will be a valid useable `vx_url_handle` for input.
- **3.7.1.6** `vx_url_write_f` will output size bytes from location ptr to the given url handle h, and return VX_SUCCESS if successful.
- **3.7.1.7** vx_url_read_f will input at most size bytes to location ptr from the given url handle h, and return VX_SUCCESS if successful. The number of bytes actually read will be returned at bytes_read.
- **3.7.1.8** vx_url_close_f will close the given url handle h, and return VX_SUCCESS if successful.
- **3.7.1.9** In all cases, a non-recoverable error is indicated when the return value is not VX_SUCCESS, and the import/export operation will be terminated.

## 3.8 Export Objects To URL

### Functions

- VX_API_ENTRY vx_status VX_API_CALL vxExportObjectsToURL (vx_context context, vx_size numrefs, const vx_reference *refs, const vx_enum *uses, const vx_char *url, vx_url_create_f create_f, vx_url_write_f write_f, vx_url_close_f close_f)

### 3.8.1 Detailed Description

Export objects to an arbitrary URL.

- **3.8.1.1** The function to export objects to an arbitrary URL is as the function documentation section.
- **3.8.1.2** Optionally, an implementation may examine the url path to decide upon what output format to use. There is no requirement to enforce compliance to the URL standard.
- **3.8.1.3** If the list of references contains one or more vx_graph, then these are first verified if necessary. (Any optimisations made by the framework should take into account which references are to be exported and how)
- **3.8.1.4** If the function is called with NULL for any of the parameters, it will return an error.
- **3.8.1.5** vxExportObjectsToURL will create a dump of the objects using the call-back functions described in the previous section, taking account of the uses specified with the uses parameter.
- **3.8.1.6** The function will call create_f once using url as the parameter, write_f zero or more times, and close_f at most once.
- **3.8.1.7** When calling write_f and close_f the url handle returned by create_f will be used.
- **3.8.1.8** If create_f returns NULL, the implementation will not call write_f or close_f.
- **3.8.1.9** If write_f returns an error, the implementation will not call write_f again, and will return without calling close_f.
- **3.8.1.10** The function will fail if any of the call-back functions fail.
- **3.8.1.11** If the export is successful (see also preceding sections), the return value is VX_SUCCESS.
- **3.8.1.12** [SC] A non-recoverable error is indicated when the return value is not VX_SUCCESS.
- **3.8.1.13** 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.
- **3.8.1.14** The reference counts of the objects listed for export will not be affected by calling this function.
- **3.8.1.15** [SC] It is a pre-requisite that all pointers passed to this function are valid. The function should attempt to validate that all pointers are valid and that the references in the given array are valid.
- **3.8.1.16** To be conformant to this extension specification an implementation need not implement this function.

### 3.8.2 Function Documentation

**VX_API_ENTRY vx_status VX_API_CALL vxExportObjectsToURL ( vx_context *context*, vx_size *numrefs*, const vx_reference * *refs*, const vx_enum * *uses*, const vx_char * *url*, vx_url_create_f *create_f*, vx_url_write_f *write_f*, vx_url_close_f *close_f* )**

Export objects to an arbitrary URL.

**Parameters**

| | | |
|---|---|---|
| in | *context* | Context from which to export objects. |
| in | *numrefs* | Number of references to export. |
| in | *refs* | References to export. |
| in | *uses* | How to export the references, use vx_export_use_e. |
| in | *url* | Url path . |
| in | *create_f* | Callback to create a url . |
| in | *write_f* | Callback to write a chunk to url . |
| in | *close_f* | Callback to close a url . |

Returns

vx_status A vx_status_e enumeration.

## 3.9 Import Objects From URL

### Functions

- VX_API_ENTRY vx_import VX_API_CALL vxImportObjectsFromURL (vx_context context, vx_size numrefs, vx_reference *refs, const vx_enum *uses, const vx_char *url, vx_url_open_f open_f, vx_url_read_f read_f, vx_url_close_f close_f)

### 3.9.1 Detailed Description

Import objects from an arbitrary URL.

- **3.9.1.1** The function to export objects to an arbitrary URL is as the function documentation section.
- **3.9.1.2** Optionally, an implementation may examine the url path to decide upon what output format to use. There is no requirement to enforce compliance to the URL standard.
- **3.9.1.3** If the function is called with NULL for any of the parameters, it will return an error.
- **3.9.1.4** A non-recoverable error is indicated when the return value is NULL, otherwise the return value is a valid vx_import. (see also preceding sections for causes of failure)
- **3.9.1.5** The import will fail if the number of references given on import does not match the number of references given on export.
- **3.9.1.6** The import will fail if the list of uses does not match the list of uses given upon export.
- **3.9.1.7** On successful import, references will be written to those entries in the array refs where the corresponding entry in the array uses is not VX_EXPORT_USE_APPLICATION_CREATE.
- **3.9.1.8** The application must create any objects marked with a use of VX_EXPORT_USE_APPLICATION_CREATE and the import will fail if such objects do not sufficiently match those given on export.
- **3.9.1.9** Releasing the import object does not automatically release the references at refs – they are not “borrowed”
- **3.9.1.10** Releasing the references at refs does not mean that the objects are deallocated, since references may also exist in the import object.
- **3.9.1.11** The function will call open_f once using url as the parameter, read_f zero or more times, and close_f at most once.
- **3.9.1.12** When calling read_f and close_f the url handle returned by open_f will be used.
- **3.9.1.13** If open_f returns NULL, the implementation will not call read_f or close_f.
- **3.9.1.14** If read_f returns an error, the implementation will not call read_f again, and will return without calling close_f.
- **3.9.1.15** If the number of bytes returned by read_f is less than the number of bytes requested, but greater than zero, the implementation may still continue to call read_f.
- **3.9.1.16** If the number of bytes returned by read_f is equal to zero, then the implementation will not call read_f again.
- **3.9.1.17** The function will fail if any of the call-back functions fail.
- **3.9.1.18** If the export is successful (see also preceding sections), the return value is VX_SUCCESS.
- **3.9.1.19** [SC] A non-recoverable error is indicated when the return value is not VX_SUCCESS.
- **3.9.1.20** 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.
- **3.9.1.21** The reference counts of the objects listed for export will not be affected by calling this function.
- **3.9.1.22** [SC] It is a pre-requisite that all pointers passed to this function are valid. The function should attempt to validate that all pointers are valid and that the references in the given array are valid.
- **3.9.1.23** To be conformant to this extension specification an implementation need not implement this function.

### 3.9.2 Function Documentation

**VX_API_ENTRY vx_import VX_API_CALL vxImportObjectsFromURL ( vx_context *context,* vx_size *numrefs,* vx_reference * *refs,* const vx_enum * *uses,* const vx_char * *url,* vx_url_open_f *open_f,* vx_url_read_f *read_f,* vx_url_close_f *close_f* )**

Import objects to an arbitrary URL.

**Parameters**

| | | |
|---|---|---|
| in | *context* | Context into which to import objects. |
| in | *numrefs* | Number of references to import. |
| in,out | *refs* | References imported. |
| in | *uses* | To import the references, use vx_export_use_e. |
| in | *url* | Url path. |
| in | *open_f* | Callback to open a url. |
| in | *read_f* | Callback to read a chunk from url. |
| in | *close_f* | Callback to close a url. |

Returns

vx_import.

# Index