GXF Core C APIs

Table of contents

Context

Create context

`gxf_result_t GxfContextCreate(gxf_context_t* context);`

Creates a new GXF context

A GXF context is required for all almost all GXF operations. The context must be destroyed with 'GxfContextDestroy'. Multiple contexts can be created in the same process, however they can not communicate with each other.

parameter: `context` The new GXF context is written to the given pointer.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Create a context from a shared context

`gxf_result_t GxfContextCreate1(gxf_context_t shared, gxf_context_t* context);`

Creates a new runtime context from shared context.

A shared runtime context is used for sharing entities between graphs running within the same process.

parameter: `shared` A valid GXF shared context.

parameter: `context` The new GXF context is written to the given pointer

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Destroy context

`gxf_result_t GxfContextDestroy(gxf_context_t context);`

Destroys a GXF context

Every GXF context must be destroyed by calling this function. The context must have been previously created with ‘GxfContextCreate’. This will also destroy all entities and components which were created as part of the context.

parameter: `context` A valid GXF context.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Extensions

Maximum number of extensions in a context can be `1024`.

Load Extensions from a file

`gxf_result_t GxfLoadExtension(gxf_context_t context, const char* filename);`

Loads extension in the given context from file.

parameter: `context` A valid GXF context

parameter: `filename` A valid filename.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

This function will be deprecated.

Load Extension libraries

```
gxf_result_t GxfLoadExtensions(gxf_context_t context, const GxfLoadExtensionsInfo*
info);
```

Loads GXF extension libraries

Loads one or more extensions either directly by their filename or indirectly by loading manifest files. Before a component can be added to a GXF entity the GXF extension shared library providing the component must be loaded. An extensions must only be loaded once.

To simplify loading multiple extensions at once the developer can create a manifest file which lists all extensions he needs. This function will then load all extensions listed in the manifest file. Multiple manifest may be loaded, however each extensions may still be loaded only a single time.

A manifest file is a YAML file with a single top-level entry 'extensions' followed by a list of filenames of GXF extension shared libraries.

Example: —– START OF FILE —– extensions: - gxf/std/libgxf_std.so - gxf/npp/libgxf_npp.so —– END OF FILE —–

parameter: `context` A valid GXF context

parameter: `filename` A valid filename.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

```
gxf_result_t GxfLoadExtensionManifest(gxf_context_t context, const char*
manifest_filename);
```

Loads extensions from manifest file.

parameter: `context` A valid GXF context.

parameter: `filename` A valid filename.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

This function will be deprecated.

Load Metadata files

```
gxf_result_t GxfLoadExtensionMetadataFiles(gxf_context_t context, const char* const*
filenames, uint32_t count);
```

Loads an extension registration metadata file

Reads a metadata file of the contents of an extension used for registration. These metadata files can be used to resolve typename and TID's of components for other extensions which depend on them. Metadata files do not contain the actual implementation of the extension and must be loaded only to run the extension query API's on extension libraries which have the actual implementation and only depend on the metadata for type resolution.

If some components of extension B depend on some components in extension A: - Load metadata file for extension A - Load extension library for extension B using 'GxfLoadExtensions' - Run extension query api's on extension B and it's components.

parameter: `context` A valid GXF context.

parameter: `filenames` absolute paths of metadata files.

parameter: `count` The number of metadata files to be loaded

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Register component

```
gxf_result_t GxfRegisterComponent(gxf_context_t context, gxf_tid_t tid, const char*
name, const char* base_name);
```

Registers a component with a GXF extension

A GXF extension need to register all of its components in the extension factory function. For convenience the helper macros in gxf/std/extension_factory_helper.hpp can be used.

The developer must choose a unique GXF tid with two random 64-bit integers. The developer must ensure that every GXF component has a unique tid. The name of the component must be the fully qualified C++ type name of the component. A component may only have a single base class and that base class must be specified with its fully qualified C++ type name as the parameter 'base_name'.

ref: gxf/std/extension_factory_helper.hpp ref: core/type_name.hpp

parameter: `context` A valid GXF context

parameter: `tid` The chosen GXF tid

parameter: `name` The type name of the component

parameter: `base_name` The type name of the base class of the component

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Graph Execution

Loads a list of entities from YAML file

```
gxf_result_t GxfGraphLoadFile(gxf_context_t context, const char* filename, const char* parameters_override[], const uint32_t num_overrides);
```

parameter: `context` A valid GXF context

parameter: `filename` A valid YAML filename.

parameter: `params_override` An optional array of strings used for override parameters in yaml file.

parameter: `num_overrides` Number of optional override parameter strings.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Set the root folder for searching YAML files during loading

`gxf_result_t GxfGraphSetRootPath(gxf_context_t context, const char* path);`

parameter: `context` A valid GXF context

parameter: `path` Path to root folder for searching YAML files during loading

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Loads a list of entities from YAML text

```
gxf_result_t GxfGraphParseString(gxf_context_t context, const char* tex, const char*
parameters_override[], const uint32_t num_overrides);
```

parameter: `context` A valid GXF context

parameter: `text` A valid YAML text.

parameter: `params_override` An optional array of strings used for override parameters in yaml file.

parameter: `num_overrides` Number of optional override parameter strings.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Activate all system components

`gxf_result_t GxfGraphActivate(gxf_context_t context);`

parameter: `context` A valid GXF context

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Deactivate all System components

`gxf_result_t GxfGraphDeactivate(gxf_context_t context);`

parameter: `context` A valid GXF context

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Starts the execution of the graph asynchronously

`gxf_result_t GxfGraphRunAsync(gxf_context_t context);`

parameter: `context` A valid GXF context

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Interrupt the execution of the graph

`gxf_result_t GxfGraphInterrupt(gxf_context_t context);`

parameter: `context` A valid GXF context

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Waits for the graph to complete execution

```
gxf_result_t GxfGraphWait(gxf_context_t context);
```

parameter: `context` A valid GXF context

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.`

Runs all System components and waits for their completion

```
gxf_result_t GxfGraphRun(gxf_context_t context);
```

parameter: `context` A valid GXF context

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Entities

Create an entity

```
gxf_result_t GxfEntityCreate(gxf_context_t context, gxf_uid_t* eid);
```

Creates a new entity and updates the eid to the unique identifier of the newly created entity.

This method will be deprecated.

```
gxf_result_t GxfCreateEntity((gxf_context_t context, const GxfEntityCreateInfo* info,
gxf_uid_t* eid);
```

Create a new GXF entity.

Entities are light-weight containers to hold components and form the basic building blocks of a GXF application. Entities are created when a GXF file is loaded, or they can be created manually using this function. Entities created with this function must be destroyed using ‘GxfEntityDestroy’. After the entity was created components can be added to it with ‘GxfComponentAdd’. To start execution of codelets on an entity the entity needs to be activated first. This can happen automatically using ‘GXF_ENTITY_CREATE_PROGRAM_BIT’ or manually using ‘GxfEntityActivate’.

parameter `context:` GXF context that creates the entity. parameter `info:` pointer to a GxfEntityCreateInfo structure containing parameters affecting the creation of the entity. parameter `eid:` pointer to a gxf_uid_t handle in which the resulting entity is returned. returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Activate an entity

`gxf_result_t GxfEntityActivate(gxf_context_t context, gxf_uid_t eid);`

Activates a previously created and inactive entity

Activating an entity generally marks the official start of its lifetime and has multiple implications: - If mandatory parameters, i.e. parameter which do not have the flag “optional”, are not set the operation will fail.

- All components on the entity are initialized.
- All codelets on the entity are scheduled for execution. The scheduler will start calling start, tick and stop functions as specified by scheduling terms.
- After activation trying to change a dynamic parameters will result in a failure.
- Adding or removing components of an entity after activation will result in a failure.

  parameter: `context` A valid GXF context

  parameter: `eid` UID of a valid entity

  returns: GXF error code

Deactivate an entity

`gxf_result_t GxfEntityDeactivate(gxf_context_t context, gxf_uid_t eid);`

Deactivates a previously activated entity

Deactivating an entity generally marks the official end of its lifetime and has multiple implications:

- All codelets are removed from the schedule. Already running entities are run to completion.
- All components on the entity are deinitialized.
- Components can be added or removed again once the entity was deactivated.
- Mandatory and non-dynamic parameters can be changed again.

Note: In case that the entity is currently executing this function will wait and block until

the current execution is finished.

parameter: `context` A valid GXF context

parameter: `eid` UID of a valid entity

returns: GXF error code

Destroy an entity

`gxf_result_t GxfEntityDestroy(gxf_context_t context, gxf_uid_t eid);`

Destroys a previously created entity

Destroys an entity immediately. The entity is destroyed even if the reference count has not yet reached 0. If the entity is active it is deactivated first.

Note: This function can block for the same reasons as ‘GxfEntityDeactivate’.

parameter: `context` A valid GXF context

parameter: `eid` The returned UID of the created entity

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Find an entity

```
gxf_result_t GxfEntityFind(gxf_context_t context, const char* name, gxf_uid_t* eid);
```

Finds an entity by its name

parameter: `context` A valid GXF context

parameter: `name` A C string with the name of the entity. Ownership is not transferred.

parameter: `eid` The returned UID of the entity

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Find all entities

```
gxf_result_t GxfEntityFindAll(gxf_context_t context, uint64_t* num_entities, gxf_uid_t*
entities);
```

Finds all entities in the current application

Finds and returns all entity ids for the current application. If more than *max_entities* exist only *max_entities* will be returned. The order and selection of entities returned is arbitrary.

parameter: `context` A valid GXF context

parameter: `num_entities` In/Out: the max number of entities that can fit in the buffer/the number of entities that exist in the application

parameter: `entities` A buffer allocated by the caller for returned UIDs of all entities, with capacity for *num_entities*.

returns: GXF_SUCCESS if the operation was successful, GXF_QUERY_NOT_ENOUGH_CAPACITY if more entities exist in the application than *max_entities*, or otherwise one of the GXF error codes.

Increase reference count of an entity

`gxf_result_t GxfEntityRefCountInc(gxf_context_t context, gxf_uid_t eid);`

Increases the reference count for an entity by 1.

By default reference counting is disabled for an entity. This means that entities created with ‘GxfEntityCreate’ are not automatically destroyed. If this function is called for an entity with disabled reference count, reference counting is enabled and the reference count is set to 1. Once reference counting is enabled an entity will be automatically destroyed if the reference count reaches zero, or if ‘GxfEntityCreate’ is called explicitly.

parameter: `context` A valid GXF context

parameter: `eid` The UID of a valid entity

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Decrease reference count of an entity

`gxf_result_t GxfEntityRefCountDec(gxf_context_t context, gxf_uid_t eid);`

Decreases the reference count for an entity by 1.

See ‘GxfEntityRefCountInc’ for more details on reference counting.

parameter: `context` A valid GXF context

parameter: `eid` The UID of a valid entity

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get status of an entity

```
gxf_result_t GxfEntityGetStatus(gxf_context_t context, gxf_uid_t eid, gxf_entity_status_t*
entity_status);
```

Gets the status of the entity.

See ‘gxf_entity_status_t’ for the various status.

parameter: `context` A valid GXF context

parameter: `eid` The UID of a valid entity

parameter: `entity_status` output; status of an entity eid

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get state of an entity

```
gxf_result_t GxfEntityGetState(gxf_context_t context, gxf_uid_t eid, entity_state_t*
entity_state);
```

Gets the state of the entity.

See ‘gxf_entity_status_t’ for the various status.

parameter: `context` A valid GXF context

parameter: `eid` The UID of a valid entity

parameter: `entity_state` output; behavior status of an entity eid used by the behavior tree parent codelet

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Notify entity of an event

`gxf_result_t GxfEntityEventNotify(gxf_context_t context, gxf_uid_t eid);`

Notifies the occurrence of an event and inform the scheduler to check the status of the entity

The entity must have an ‘AsynchronousSchedulingTerm’ scheduling term component and it must be in “EVENT_WAITING” state for the notification to be acknowledged.

See ‘AsynchronousEventState’ for various states

parameter: `context` A valid GXF context

parameter: `eid` The UID of a valid entity

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Components

Maximum number of components in an entity or an extension can be up to `1024`.

Get component type identifier

`gxf_result_t GxfComponentTypeId(gxf_context_t context, const char* name, gxf_tid_t* tid);`

Gets the GXF unique type ID (TID) of a component

Get the unique type ID which was used to register the component with GXF. The function expects the fully qualified C++ type name of the component including namespaces.

Example of a valid component type name: “nvidia::gxf::test::PingTx”

parameter: `context` A valid GXF context

parameter: `name` The fully qualified C++ type name of the component

parameter: `tid` The returned TID of the component

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get component type name

```
gxf_result_t GxfComponentTypeName(gxf_context_t context, gxf_tid_t tid, const char** name);
```

Gets the fully qualified C++ type name GXF component typename

Get the unique typename of the component with which it was registered using one of the GXF_EXT_FACTORY_ADD*() macros

parameter: `context` A valid GXF context

parameter: `tid` The unique type ID (TID) of the component with which the component was registered

parameter: `name` The returned name of the component

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get component name

```
gxf_result_t GxfComponentName(gxf_context_t context, gxf_uid_t cid, const char** name);
```

Gets the name of a component

Each component has a user-defined name which was used in the call to ‘GxfComponentAdd’. Usually the name is specified in the GXF application file.

parameter: `context` A valid GXF context

parameter: `cid` The unique object ID (UID) of the component

parameter: `name` The returned name of the component

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get unique identifier of the entity of given component

```
gxf_result_t GxfComponentEntity(gxf_context_t context, gxf_uid_t cid, gxf_uid_t* eid);
```

Gets the unique object ID of the entity of a component

Each component has a unique ID with respect to the context and is stored in one entity. This function can be used to retrieve the ID of the entity to which a given component belongs.

parameter: `context` A valid GXF context

parameter: `cid` The unique object ID (UID) of the component

parameter: `eid` The returned UID of the entity

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Add a new component

```
gxf_result_t GxfComponentAdd(gxf_context_t context, gxf_uid_t eid, gxf_tid_t tid, const
char* name, gxf_uid_t* cid);
```

Adds a new component to an entity

An entity can contain multiple components and this function can be used to add a new component to an entity. A component must be added before an entity is activated, or after it was deactivated. Components must not be added to active entities. The order of components is stable and identical to the order in which components are added (see ‘GxfComponentFind’).

parameter: `context` A valid GXF context

parameter: `eid` The unique object ID (UID) of the entity to which the component is added.

parameter: `tid` The unique type ID (TID) of the component to be added to the entity.

parameter: `name` The name of the new component. Ownership is not transferred.

parameter: `cid` The returned UID of the created component

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Add component to entity interface

```
gxf_result_t GxfComponentAddToInterface(gxf_context_t context, gxf_uid_t eid, gxf_uid_t cid, const char* name);
```

Adds an existing component to the interface of an entity

An entity can holds references to other components in its interface, so that when finding a component in an entity, both the component this entity holds and those it refers to will be returned. This supports the case when an entity contains a subgraph, then those components that has been declared in the subgraph interface will be put to the interface of the parent entity.

parameter: `context` A valid GXF context

parameter: `eid` The unique object ID (UID) of the entity to which the component is added.

parameter: `cid` The unique object ID of the component.

parameter: `name` The name of the new component. Ownership is not transferred.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Find a component in an entity

```
gxf_result_t GxfComponentFind(gxf_context_t context, gxf_uid_t eid, gxf_tid_t tid, const char* name, int32_t* offset, gxf_uid_t* cid);
```

Finds a component in an entity

Searches components in an entity which satisfy certain criteria: component type, component name, and component min index. All three criteria are optional; in case no criteria is given the first component is returned. The main use case for "component min index" is a repeated search which continues at the index which was returned by a previous search.

In case no entity with the given criteria was found GXF_ENTITY_NOT_FOUND is returned.

parameter: `context` A valid GXF context

parameter: `eid` The unique object ID (UID) of the entity which is searched.

parameter: `tid` The component type ID (TID) of the component to find (optional)

parameter: `name` The component name of the component to find (optional). Ownership not transferred.

parameter: `offset` The index of the first component in the entity to search. Also contains the index of the component which was found.

parameter: `cid` The returned UID of the searched component

returns: GXF_SUCCESS if a component matching the criteria was found, GXF_ENTITY_NOT_FOUND if no component matching the criteria was found, or otherwise one of the GXF error codes.

Get type identifier for a component

```
gxf_result_t GxfComponentType(gxf_context_t context, gxf_uid_t cid, gxf_tid_t* tid);
```

Gets the component type ID (TID) of a component

parameter: `context` A valid GXF context

parameter: `cid` The component object ID (UID) for which the component type is requested.

parameter: `tid` The returned TID of the component

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Gets pointer to component

```
gxf_result_t GxfComponentPointer(gxf_context_t context, gxf_uid_t uid, gxf_tid_t tid,
void** pointer);
```

Verifies that a component exists, has the given type, gets a pointer to it.

parameter: `context` A valid GXF context

parameter: `uid` The component object ID (UID).

parameter: `tid` The expected component type ID (TID) of the component

parameter: `pointer` The returned pointer to the component object.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Primitive Parameters

64-bit floating point

Set

```
gxf_result_t GxfParameterSetFloat64(gxf_context_t context, gxf_uid_t uid, const char* key, double value);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` a double value

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get

```
gxf_result_t GxfParameterGetFloat64(gxf_context_t context, gxf_uid_t uid, const char* key, double* value);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` pointer to get the double value.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

64-bit signed integer

Set

```
gxf_result_t GxfParameterSetInt64(gxf_context_t context, gxf_uid_t uid, const char* key,
int64_t value);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` 64-bit integer value to set.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get

```
gxf_result_t GxfParameterGetInt64(gxf_context_t context, gxf_uid_t uid, const char* key,
int64_t* value);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` pointer to get the 64-bit integer value.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

64-bit unsigned integer

Set

```
gxf_result_t GxfParameterSetUInt64(gxf_context_t context, gxf_uid_t uid, const char*
key, uint64_t value);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` unsigned 64-bit integer value to set.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get

```
gxf_result_t GxfParameterGetUInt64(gxf_context_t context, gxf_uid_t uid, const char*
key, uint64_t* value);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` pointer to get the unsigned 64-bit integer value.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

32-bit signed integer

Set

```
gxf_result_t GxfParameterSetInt32(gxf_context_t context, gxf_uid_t uid, const char* key,
int32_t value);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` 32-bit integer value to set.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get

```
gxf_result_t GxfParameterGetInt32(gxf_context_t context, gxf_uid_t uid, const char* key,
int32_t* value);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` pointer to get the 32-bit integer value.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

String parameter

Set

```
gxf_result_t GxfParameterSetStr(gxf_context_t context, gxf_uid_t uid, const char* key,
const char* value);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` A char array containing value to set.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get

```
gxf_result_t GxfParameterGetStr(gxf_context_t context, gxf_uid_t uid, const char* key,
const char** value);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` pointer to a char* array to get the value.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Boolean

Set

```
gxf_result_t GxfParameterSetBool(gxf_context_t context, gxf_uid_t uid, const char* key,
bool value);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` A boolean value to set.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get

```
gxf_result_t GxfParameterGetBool(gxf_context_t context, gxf_uid_t uid, const char* key, bool* value);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` pointer to get the boolean value.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Handle

Set

```
gxf_result_t GxfParameterSetHandle(gxf_context_t context, gxf_uid_t uid, const char* key, gxf_uid_t cid);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `cid` Unique identifier to set.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get

```
gxf_result_t GxfParameterGetHandle(gxf_context_t context, gxf_uid_t uid, const char*
key, gxf_uid_t* cid);
```

parameter: `context` A valid GXF context.

parameter: `uid` A valid component identifier.

parameter: `key` A valid name of a component to set.

parameter: `value` Pointer to a unique identifier to get the value.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Vector Parameters

To set or get the vector parameters of a component, users can use the following C-APIs for various data types:

Set 1-D Vector Parameters

Users can call `gxf_result_t GxfParameterSet1D"DataType"Vector(gxf_context_t context, gxf_uid_t uid, const char* key, data_type* value, uint64_t length)`

`value` should point to an array of the data to be set of the corresponding type. The size of the stored array should match the `length` argument passed.

See the table below for all the supported data types and their corresponding function signatures.

parameter: `key` The name of the parameter

parameter: `value` The value to set of the parameter

parameter: `length` The length of the vector parameter

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Table 1 *Supported Data Types to Set 1D Vector Parameters*

Function Name	data_type
`GxfParameterSet1DFloat64Vector(...)`	`double`
`GxfParameterSet1DInt64Vector(...)`	`int64_t`
`GxfParameterSet1DUInt64Vector(...)`	`uint64_t`
`GxfParameterSet1DInt32Vector(...)`	`int32_t`

Set 2-D Vector Parameters

Users can call `gxf_result_t GxfParameterSet2D"DataType"Vector(gxf_context_t context, gxf_uid_t uid, const char* key, data_type** value, uint64_t height, uint64_t width)`

`value` should point to an array of array (and not to the address of a contiguous array of data) of the data to be set of the corresponding type. The length of the first dimension of the array should match the `height` argument passed and similarly the length of the second dimension of the array should match the `width` passed.

See the table below for all the supported data types and their corresponding function signatures.

parameter: `key` The name of the parameter

parameter: `value` The value to set of the parameter

parameter: `height` The height of the 2-D vector parameter

parameter: `width` The width of the 2-D vector parameter

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Table 2 *Supported Data Types to Set 2D Vector Parameters*

Function Name	data_type
`GxfParameterSet2DFloat64Vector(...)`	`double`
`GxfParameterSet2DInt64Vector(...)`	`int64_t`
`GxfParameterSet2DUInt64Vector(...)`	`uint64_t`
`GxfParameterSet2DInt32Vector(...)`	`int32_t`

Get 1-D Vector Parameters

Users can call `gxf_result_t GxfParameterGet1D"DataType"Vector(gxf_context_t context, gxf_uid_t uid, const char* key, data_type** value, uint64_t* length)` to get the value of a 1-D vector.

Before calling this method, users should call

```
GxfParameterGet1D"DataType"VectorInfo(gxf_context_t context, gxf_uid_t uid, const char* key, uint64_t* length)
```

to obtain the `length` of the vector parameter and then allocate at least that much memory to retrieve the `value`.

`value` should point to an array of size greater than or equal to `length` allocated by user of the corresponding type to retrieve the data. If the `length` doesn’t match the size of stored vector then it will be updated with the expected size.

See the table below for all the supported data types and their corresponding function signatures.

parameter: `key` The name of the parameter

parameter: `value` The value to set of the parameter

parameter: `length` The length of the 1-D vector parameter obtained by calling `GxfParameterGet1D"DataType"VectorInfo(...)`

Table 3 *Supported Data Types to Get the Value of 1D Vector Parameters*

Function Name	data_type
`GxfParameterGet1DFloat64Vector(...)`	`double`
`GxfParameterGet1DInt64Vector(...)`	`int64_t`
`GxfParameterGet1DUInt64Vector(...)`	`uint64_t`
`GxfParameterGet1DInt32Vector(...)`	`int32_t`

Get 2-D Vector Parameters

Users can call `gxf_result_t GxfParameterGet2D"DataType"Vector(gxf_context_t context, gxf_uid_t uid, const char* key, data_type** value, uint64_t* height, uint64_t* width)` to get the value of a -2D vector.

Before calling this method, users should call

```
GxfParameterGet1D"DataType"VectorInfo(gxf_context_t context, gxf_uid_t uid, const
char* key, uint64_t* height, uint64_t* width)
```

to obtain the `height` and `width` of the 2D-vector parameter and then allocate at least that much memory to retrieve the `value`.

`value` should point to an array of array of height (size of first dimension) greater than or equal to `height` and width (size of the second dimension) greater than or equal to `width` allocated by user of the corresponding type to get the data. If the `height` or `width` don't match the height and width of the stored vector then they will be updated with the expected values.

See the table below for all the supported data types and their corresponding function signatures.

parameter": `key` The name of the parameter

parameter": `value` Allocated array to get the value of the parameter

parameter”: `height` The height of the 2-D vector parameter obtained by calling `GxfParameterGet2D"DataType"VectorInfo(...)`

parameter”: `width` The width of the 2-D vector parameter obtained by calling `GxfParameterGet2D"DataType"VectorInfo(...)`

Table 4 *Supported Data Types to Get the Value of 2D Vector Parameters*

Function Name	data_type
`GxfParameterGet2DFloat64Vector(...)`	`double`
`GxfParameterGet2DInt64Vector(...)`	`int64_t`
`GxfParameterGet2DUInt64Vector(...)`	`uint64_t`
`GxfParameterGet2DInt32Vector(...)`	`int32_t`

Information Queries

Get Meta Data about the GXF Runtime

```
gxf_result_t GxfRuntimeInfo(gxf_context_t context, gxf_runtime_info* info);
```

parameter: `context` A valid GXF context.

parameter: `info` pointer to gxf_runtime_info object to get the meta data.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get description and list of components in loaded Extension

```
gxf_result_t GxfExtensionInfo(gxf_context_t context, gxf_tid_t tid, gxf_extension_info_t*
info);
```

parameter: `context` A valid GXF context.

parameter: `tid` The unique identifier of the extension.

parameter: `info` pointer to gxf_extension_info_t object to get the meta data.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get description and list of parameters of Component

```
gxf_result_t GxfComponentInfo(gxf_context_t context, gxf_tid_t tid,
gxf_component_info_t* info);
```

Note: Parameters are only available after at least one instance is created for the Component.

parameter: `context` A valid GXF context.

parameter: `tid` The unique identifier of the component.

parameter: `info` pointer to gxf_component_info_t object to get the meta data.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Get parameter type description

Gets a string describing the parameter type

```
const char* GxfParameterTypeStr(gxf_parameter_type_t param_type);
```

parameter: `param_type` Type of parameter to get info about.

returns: C-style string description of the parameter type.

Get flag type description

Gets a string describing the flag type

`const char* GxfParameterFlagTypeStr(gxf_parameter_flags_t_ flag_type);`

parameter: `flag_type` Type of flag to get info about.

returns: C-style string description of the flag type.

Get parameter description

Gets description of specific parameter. Fails if the component is not instantiated yet.

```
gxf_result_t GxfGetParameterInfo(gxf_context_t context, gxf_tid_t cid, const char* key,
gxf_parameter_info_t* info);
```

parameter: `context` A valid GXF context.

parameter: `cid` The unique identifier of the component.

parameter: `key` The name of the parameter.

parameter: `info` Pointer to a gxf_parameter_info_t object to get the value.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Redirect logs to a file

Redirect console logs to the provided file.

`gxf_result_t GxfGetParameterInfo(gxf_context_t context, FILE* fp);`

parameter: `context` A valid GXF context.

parameter: `fp` File path for the redirected logs.

returns: GXF_SUCCESS if the operation was successful, or otherwise one of the GXF error codes.

Miscellaneous

Get string description of error

`const char* GxfResultStr(gxf_result_t result);`

Gets a string describing an GXF error code.

The caller does not get ownership of the return C string and must not delete it.

parameter: `result` A GXF error code

returns: A pointer to a C string with the error code description.