



Level Zero Specification & How to Participate

Introduction

- Level Zero is foundational component of oneAPI
- Spec v1.4 published in May
 - Extensions for querying device properties and topologies
 - Spec clarifications
- Starting with Level Zero Spec v1.5
 - Support better community engagement
 - More visibility into the development of the spec
 - Open source the spec development framework (June)

<https://github.com/oneapi-src/level-zero-spec>

Level Zero Specification - Chapters

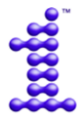
- Introduction
- Programming Guides
 - Core (ze)
 - Tools (zet)
 - Sysman (zes)
 - SPIR-V
- Extensions
 - Standard
 - Experimental
- API Documentation
- Versions

Level Zero Specification – Release Notes

Level Zero		
Version	Date	View
Level Zero v1.4	2022-05-05	HTML
Level Zero v1.3	2021-11-27	HTML
Level Zero v1.2	2021-05-11	HTML
Level Zero v1.1	2021-02-04	HTML
Level Zero v1.0	2020-10-02	HTML
Level Zero v0.95	2020-05-28	HTML
Level Zero v0.91	2020-03-04	HTML

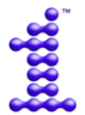
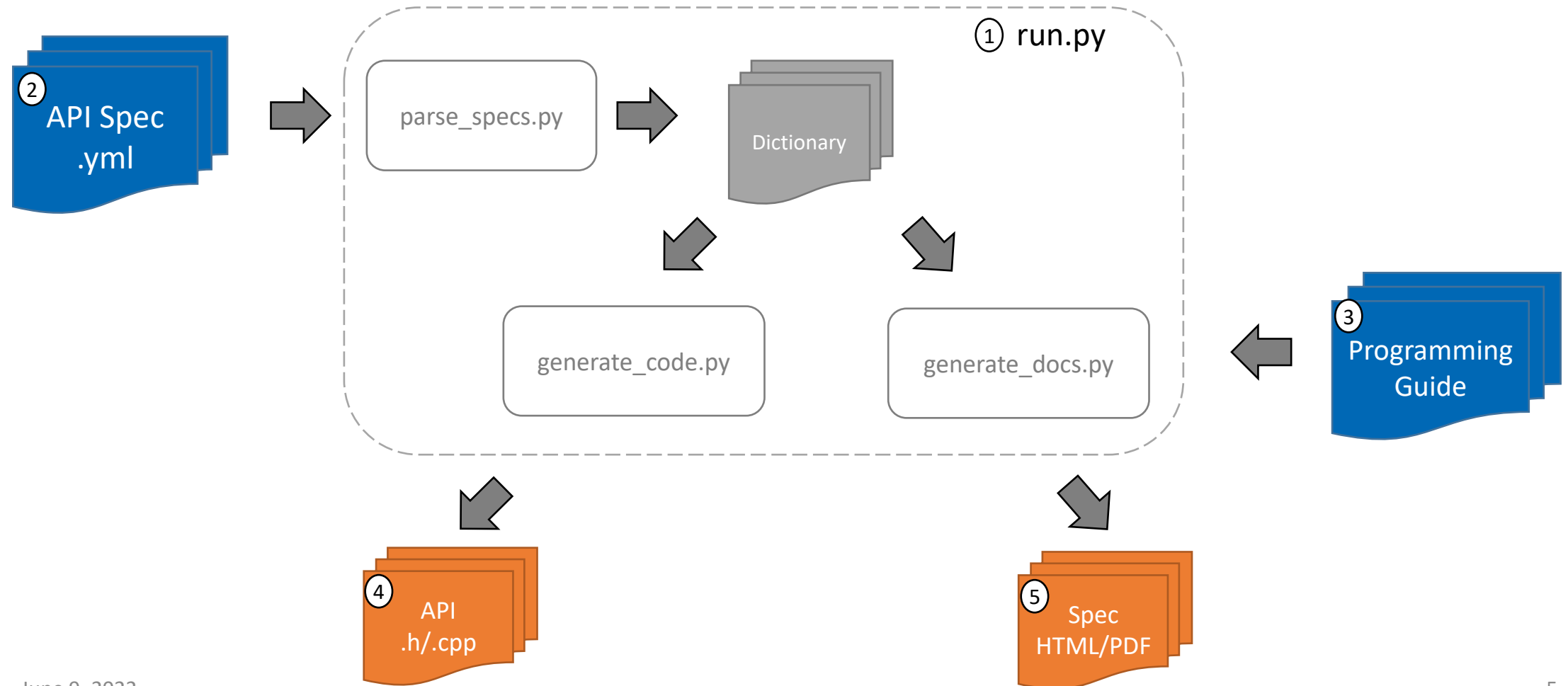
Release Notes	
Level Zero v1.4	
<ul style="list-style-type: none">• Core Changes<ul style="list-style-type: none">◦ Fabric Topology Discovery API extension added.◦ Add detail to allocation access capabilities◦ Add an extension to the Core API for obtaining memory BW◦ Add clarifications for printf◦ Add extension for querying device locally unique identifier◦ Fix reordering of stypes◦ Standardize use of desc in SetEccState	

<https://spec.oneapi.io/releases/index.html#level-zero>



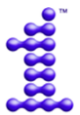
Level Zero Specification - Framework

- Scripts generate specification and headers



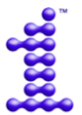
Level Zero Specification - YAML

```
--- #
type: function
desc: "Creates a context for the driver."
class: $xContext
name: Create
decl: static
ordinal: "0"
details:
  - "The application must only use the context for the driver which was"
  - "The application may call this function from simultaneous threads."
  - "The implementation of this function must be thread-safe."
params:
  - type: $x_driver_handle_t
    name: hDriver
    desc: "[in] handle of the driver object"
  - type: "const $x_context_desc_t*"
    name: desc
    desc: "[in] pointer to context descriptor"
  - type: $x_context_handle_t*
    name: phContext
    desc: "[out] pointer to handle of context object created"
returns:
  - $X_RESULT_ERROR_OUT_OF_HOST_MEMORY
  - $X_RESULT_ERROR_OUT_OF_DEVICE_MEMORY
```



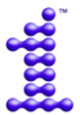
Level Zero Specification - Headers

```
////////////////////////////////////  
/// @brief Creates a context for the driver.  
///  
/// @details  
/// ..... - The application must only use the context for the driver which was  
/// ..... provided during creation.  
/// ..... - The application may call this function from simultaneous threads.  
/// ..... - The implementation of this function must be thread-safe.  
///  
/// @returns  
/// ..... - ::ZE_RESULT_SUCCESS  
/// ..... - ::ZE_RESULT_ERROR_UNINITIALIZED  
/// ..... - ::ZE_RESULT_ERROR_DEVICE_LOST  
/// ..... - ::ZE_RESULT_ERROR_INVALID_NULL_HANDLE  
/// ..... + `nullptr == hDriver`  
/// ..... - ::ZE_RESULT_ERROR_INVALID_NULL_POINTER  
/// ..... + `nullptr == desc`  
/// ..... + `nullptr == phContext`  
/// ..... - ::ZE_RESULT_ERROR_INVALID_ENUMERATION  
/// ..... + `0x1 < desc->flags`  
/// ..... - ::ZE_RESULT_ERROR_OUT_OF_HOST_MEMORY  
/// ..... - ::ZE_RESULT_ERROR_OUT_OF_DEVICE_MEMORY  
ZE_APIEXPORT ze_result_t ZE_APICALL  
zeContextCreate(  
    ..... ze_driver_handle_t hDriver, ..... ///< [in] handle of the driver  
    ..... const ze_context_desc_t* desc, ..... ///< [in] pointer to context description  
    ..... ze_context_handle_t* phContext ..... ///< [out] pointer to handle  
    ..... );
```



Level Zero Specification – Implementation

```
////////////////////////////////////  
/// @brief Creates a context for the driver.  
///  
/// @details  
/// ..... - The application must only use the context for the driver which was  
/// ..... provided during creation.  
/// ..... - The application may call this function from simultaneous threads.  
/// ..... - The implementation of this function must be thread-safe.  
///  
/// @returns  
/// ..... - ::ZE_RESULT_SUCCESS  
/// ..... - ::ZE_RESULT_ERROR_UNINITIALIZED  
/// ..... - ::ZE_RESULT_ERROR_DEVICE_LOST  
/// ..... - ::ZE_RESULT_ERROR_INVALID_NULL_HANDLE  
/// ..... + `nullptr == hDriver`  
/// ..... - ::ZE_RESULT_ERROR_INVALID_NULL_POINTER  
/// ..... + `nullptr == desc`  
/// ..... + `nullptr == phContext`  
/// ..... - ::ZE_RESULT_ERROR_INVALID_ENUMERATION  
/// ..... + `0x1 < desc->flags`  
/// ..... - ::ZE_RESULT_ERROR_OUT_OF_HOST_MEMORY  
/// ..... - ::ZE_RESULT_ERROR_OUT_OF_DEVICE_MEMORY  
ze_result_t ZE_APICALL  
zeContextCreate(  
    ..... ze_driver_handle_t hDriver, ..... ///< [in] handle of the driv  
    ..... const ze_context_desc_t* desc, ..... ///< [in] pointer to context  
    ..... ze_context_handle_t* phContext ..... ///< [out] pointer to handle  
    ..... )  
{  
    ..... ze_result_t result = ZE_RESULT_SUCCESS;  
    ..... return result;  
}
```



Level Zero Specification – API Specification

🏠 Level Zero Specification

1.4.3

Search docs

Introduction

Core Programming Guide

Tools Programming Guide

Sysman Programming Guide

SPIR-V Programming Guide

Extensions

📖 API Documentation

Versions

zeContextCreate

```
ZE_APIEXPORT ze_result_t ZE_APICALL zeContextCreate(ze_driver_handle_t hDriver, const ze_context_desc_t *desc, ze_context_handle_t *phContext)
```

Creates a context for the driver.

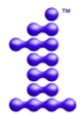
- The application must only use the context for the driver which was provided during creation.
- The application may call this function from simultaneous threads.
- The implementation of this function must be thread-safe.

Parameters:

- **hDriver** – [in] handle of the driver object
- **desc** – [in] pointer to context descriptor
- **phContext** – [out] pointer to handle of context object created

Returns:

- **ZE_RESULT_SUCCESS**
- **ZE_RESULT_ERROR_UNINITIALIZED**
- **ZE_RESULT_ERROR_DEVICE_LOST**
- **ZE_RESULT_ERROR_INVALID_NULL_HANDLE**
 - `nullptr == hDriver`
- **ZE_RESULT_ERROR_INVALID_NULL_POINTER**
 - `nullptr == desc`
 - `nullptr == phContext`
- **ZE_RESULT_ERROR_INVALID_ENUMERATION**
 - `0x1 < desc->flags`
- **ZE_RESULT_ERROR_OUT_OF_HOST_MEMORY**
- **ZE_RESULT_ERROR_OUT_OF_DEVICE_MEMORY**



Level Zero Specification – Programming Guides

🏠 Level Zero Specification
1.4.3

Search docs

Introduction

☐ Core Programming Guide

⊕ Drivers and Devices

Contexts

⊕ Memory and Images

⊕ Command Queues and Command Lists

⊕ Synchronization Primitives

⊕ Barriers

⊕ Modules and Kernels

⊕ Advanced

Tools Programming Guide

Sysman Programming Guide

SPIR-V Programming Guide

Extensions

API Documentation

Versions

Contexts

A context is a logical object used by the driver for managing all memory, command queues/lists, modules, synchronization objects, etc.

- A context handle is primarily used during creation and management of resources that may be used by multiple devices.
- For example, memory is not implicitly shared across all devices supported by a driver. However, it is available to be explicitly shared.

The following pseudo-code demonstrates a basic context creation:

```
// Create context
ze_context_desc_t ctxtDesc = {
    ZE_STRUCTURE_TYPE_CONTEXT_DESC,
    nullptr,
    0
};
zeContextCreate(hDriver, &ctxtDesc, &hContext);
```

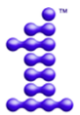
An application may optionally create multiple contexts using `zeContextCreate`.

- The primary usage-model for multiple contexts is isolation of memory and objects for multiple libraries within the same process.
- The same context may be used simultaneously on multiple Host threads.

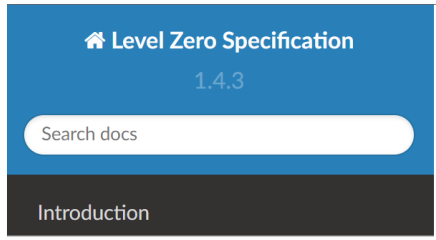
The following pseudo-code demonstrates a basic context creation and activation sequence:

```
// Create context(s)
zeContextCreate(hDriver, &ctxtDesc, &hContextA);
zeContextCreate(hDriver, &ctxtDesc, &hContextB);

zeMemAllocHost(hContextA, &desc, 80, 0, &ptrA);
```



Level Zero Specification – Programming Guides



Contexts

A context is a logical object used by the driver for managing all memory, command queues/lists, modules, synchronization objects, etc.

- A context handle is primarily used during creation and management of resources that may be

PROG.rst(122) : error : \$xContextCreate parameter count mismatch - 2 actual vs. 3 expected
Line 122 = \${x}ContextCreate(hDriver, &ctxtDesc);

- ⊕ Command Queues and Command Lists
- ⊕ Synchronization Primitives
- ⊕ Barriers
- ⊕ Modules and Kernels
- ⊕ Advanced
- Tools Programming Guide
- Sysman Programming Guide
- SPIR-V Programming Guide
- Extensions
- API Documentation
- Versions

```
// Create context
ze_context_desc_t ctxtDesc = {
    ZE_STRUCTURE_TYPE_CONTEXT_DESC,
    nullptr,
    0
};
zeContextCreate(hDriver, &ctxtDesc, &hContext);
```

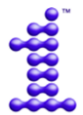
An application may optionally create multiple contexts using `zeContextCreate`.

- The primary usage-model for multiple contexts is isolation of memory and objects for multiple libraries within the same process.
- The same context may be used simultaneously on multiple Host threads.

The following pseudo-code demonstrates a basic context creation and activation sequence:

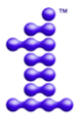
```
// Create context(s)
zeContextCreate(hDriver, &ctxtDesc, &hContextA);
zeContextCreate(hDriver, &ctxtDesc, &hContextB);

zeMemAllocHost(hContextA, &desc, 80, 0, &ptrA);
```



Level Zero Specification – YAML Versioning

```
--- #-----
type: function
desc: "Create image view on the context."
version: "1.2"
class: $xImage
name: ViewCreateExp
decl: static
ordinal: "0"
details:
  - "The application must only use the image view for the device, or its sub-"
  - "The application may call this function from simultaneous threads."
  - "The implementation of this function must be thread-safe."
  - "The implementation must support $X_experimental_image_view extension."
  - "Image views are treated as images from the API."
  - "Image views provide a mechanism to redescribe how an image is interpreted."
  - "Image views become disabled when their corresponding image resource is destroyed."
  - "Use $xImageDestroy to destroy image view objects."
analogue:
```



Versioning - Backward Compatibility

- Minor version increment
 - Additional functionality
 - Promoted extensions
 - Must retain backward compatibility
 - Adding new functions and structures
 - Adding enumerations, and using reserved bits
- Major version increment
 - Modified or remove functionality
 - May break backward compatibility
 - Modifying existing functions and structures
 - Removing functions and structures

Next Steps

- Release spec development framework
- Post spec issues from internal repo
- Organize candidate spec update for Spec v1.5

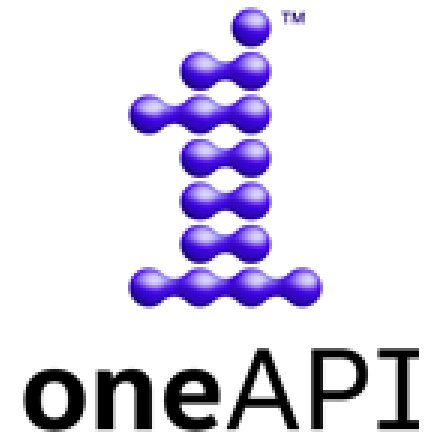
<https://github.com/oneapi-src/level-zero-spec>

Level Zero Specification and Repositories

- Level Zero Specification
 - <https://spec.oneapi.com/versions/latest/elements/l0/source/index.html>
- Level Zero Specification Github
 - <https://github.com/oneapi-src/level-zero-spec>
- Level Zero Loader
 - <https://github.com/oneapi-src/level-zero>
- Level Zero Tests (Conformance and Performance)
 - <https://github.com/oneapi-src/level-zero-tests>
- Level Zero Intel GPU Driver
 - <https://github.com/intel/compute-runtime>

Call to Action

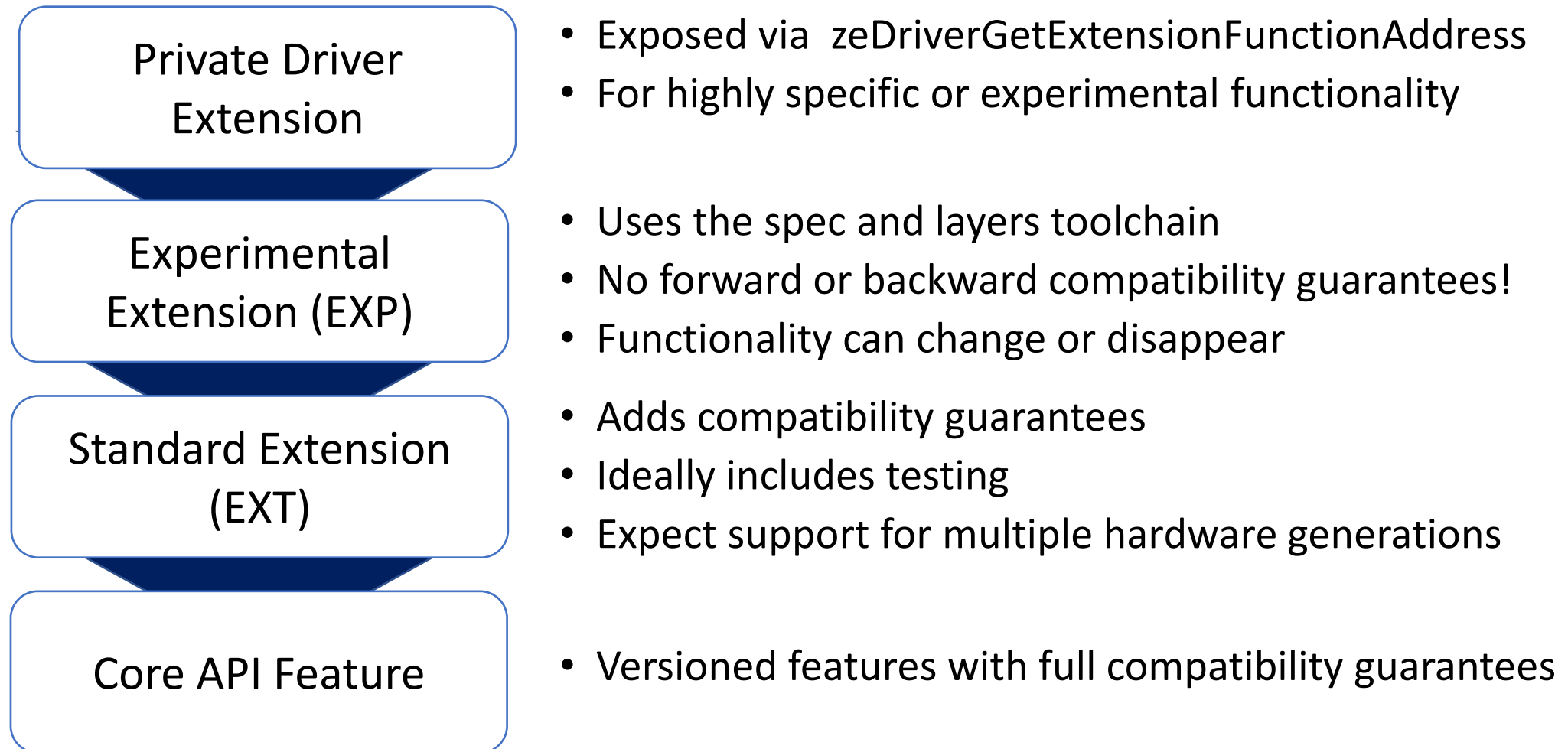
- Contribute to Level Zero Specification
 - Filing issues and contribute to discussions in GitHub
 - Contribute spec changes and extensions
- Help us evolve Level Zero
 - What new Level Zero features should we consider?
 - Level Zero clarifications?
 - What refactoring should we do for Spec 2.0?
- What Level Zero topics are interesting for future TAB meetings?



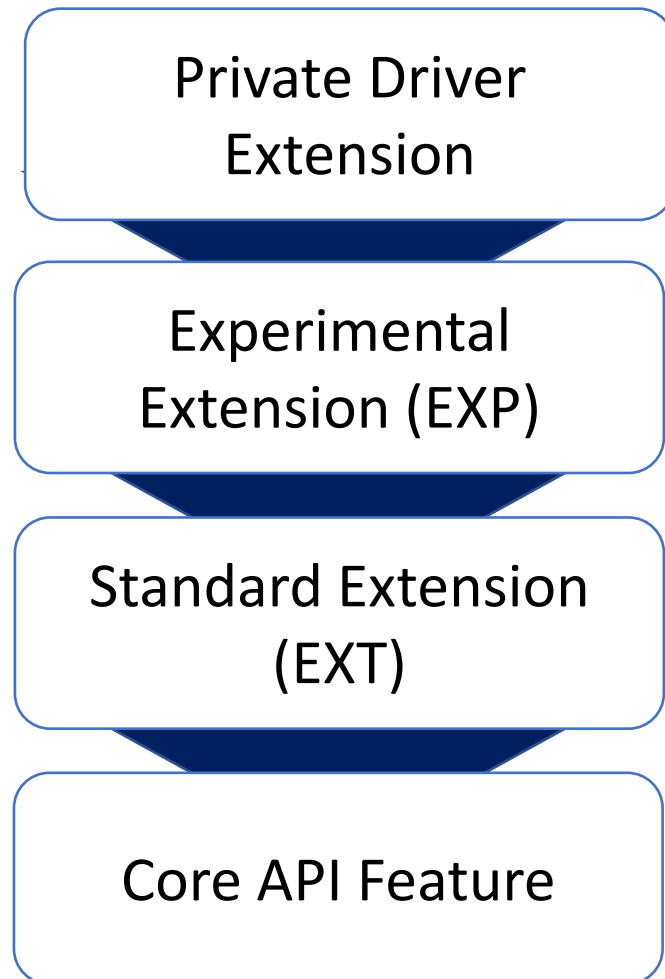
Thank You!

<http://oneapi.io>

Level Zero Extension Life Cycle



Level Zero Extension Life Cycle Notes



- All extensions may be optionally supported!
- Can skip steps, e.g. some features may be implemented directly as standard extensions or core features
- Some features may remain extensions indefinitely
- Experimental extensions should (ideally) be short-lived
 - Either dropped or transitioned to standard extensions