identifier/identifier

Release stable

Ben Ramsey

2022-11-28

CONTENTS

Version	stable
Updated	2022-11-28
Authors	B. Ramsey

Contents

CHAPTER

ONE

ABSTRACT

This specification defines common interfaces for representing and creating identifiers in PHP. Identifiers are names that identify specific objects or types of objects. Software engineers have defined many different identifiers based on a variety of needs, such as uniqueness, authority, and scalability. Some popular identifiers defined as a result of these efforts include [UUID], [ULID], and [Snowflake] identifiers.

Final implementations MAY decorate identifiers with more functionality than proposed here, but they MUST implement the indicated interfaces/functionality described in this document.

CHAPTER

TWO

INTRODUCTION

As programmers, we use identifiers everywhere. Without them, we wouldn't be able to keep track of, for example, a user's account or the location of a web page. To identify a user's account, we might use an auto-incrementing integer created by a database; for a web page, we might use a URL. Both can uniquely identify the object they refer to, based on their context—within a specific database table, the user's identifier can refer only to them, and on the web, a URL can refer only to a single web page.

Identifiers come in many forms. We've mentioned two such forms: integers and strings. Within these forms, identifiers might have metadata embedded within them. From the URL, we can assume that it resolves to some information stored on the web; we can determine the protocol, host name, and port number used to access this information. Depending on the combination of characters in the URL, we might also be able to determine the expected language of the information (e.g., if the URL includes a path segment like `/en-US/`) or the title of the page or the date of the article, if the URL includes this information in a human-readable form. On the other hand, the user's account identifier might not include any obvious metadata at all. It is a simple integer, with no context.

We can decipher information from the URL because there is a specification for how to form them. Other data included in URLs might be intuited from context clues—if part of the URL appears to have a formatted date-time string, then that date is likely related to the content at that URL, in some way.

Other types of identifiers might include non-obvious metadata that we can only decipher if we know how they were created. If we know the user's account identifier is a [Snowflake], then we know we can extract the time it was created and possibly other information, such as the machine or worker that created it.

Most identifiers share a handful of common traits, and we tend to use them similarly, despite how different they might appear on the surface. We might compare them for equality or sorting. We might convert them to a binary representation for storage or extract the date to find when they were created. This specification attempts to capture these commonalities, providing applications and libraries a means to create and pass identifiers, without being locked-in to a specific implementation.

CHAPTER

THREE

TERMINOLOGY

3.1 Requirements Language

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3.2 Definitions

Binary A base-2 numeral system, where each position has a value of either zero (0) or one (1), also known as a *bit*. In the context of this specification, binary data refers to a byte string.

Byte string A sequence of bytes in which each byte is a sequence of 8 bits, with 256 possible values. A byte string is sometimes referred to as an *octet stream*.

Date-time A representation of a point in history as a date and time, usually using an instance of `\DateTimeInterface` in PHP.

Identifier “[A] name that identifies (that is, labels the identity of) either a unique object or a unique class of objects, where the ‘object’ or class may be an idea, physical countable object (or class thereof), or physical noncountable substance (or class thereof)” [Wikipedia].

CHAPTER

FOUR

USAGE

4.1 Creating Identifiers

Applications SHOULD use factories to create identifiers, wherever possible.

```
$identifier = $factory->create();
```

4.2 Sorting Identifiers

This specification supports in-application sorting of identifiers. For example:

```
usort($identifiers, fn (Identifier $a, Identifier $b): int => $a->compareTo($b));
```

4.3 Get the Timestamp

`DateTimeIdentifier` instances support extracting the date-time value from the identifier. The return value of `getDateTime()` is a PHP `\DateTimeImmutable` instance.

```
$timestamp = $identifier->getDateTime()->getTimestamp();
```

CHAPTER

FIVE

INTERFACES

5.1 Identifiers

5.1.1 Identifier

The identifier interface defines the most basic operations all identifiers support. These basic operations provide the ability to compare identifiers and convert the identifier to a string for presentation.

To provide comparison, implementations MUST implement `compareTo()`, which MUST accept any value for comparison, returning an integer less than, equal to, or greater than zero, if the identifier is less than, equal to, or greater than the value. This allows for sorting identifiers.

Likewise, `equals()` MUST accept any value for comparison and return boolean `true` if the identifier is equal to the value and `false` otherwise.

If an implementation wishes to indicate that it cannot compare the identifier to the provided value, it MUST throw an `\Identifier\Exception\NotComparable` exception.

The `toString()` method SHOULD return a human-readable representation of the identifier (e.g., integer string, UUID, hexadecimal-encoded, Base64-encoded, etc.).

```php
namespace Identifier;

interface Identifier
{
    /**
     * Returns an integer less than, equal to, or greater than zero if the
     * identifier is less than, equal to, or greater than the other value
     *
     * @throws Exception\NotComparable MUST throw if the implementation is
     *     unable to make comparisons with the other value provided
     */
    public function compareTo(mixed $other): int;

    /**
     * Returns true if the identifier is equal to the other value
     *
     * @throws Exception\NotComparable MUST throw if the implementation is
     *     unable to make comparisons with the other value provided
     */
    public function equals(mixed $other): bool;
```

(continues on next page)

(continued from previous page)

```
    /**
     * Returns a string representation of the identifier
     */
    public function toString(): string;
}
```

5.1.2 BinaryIdentifier

A binary identifier is an identifier that is based on arbitrary binary data. The `toBytes()` method MUST return the byte string representation of the identifier.

Binary identifiers are useful especially when the size of the identifier overflows the system limitations for maximum or minimum integers. For example, UUIDs and ULIDs, both 128-bit integers, may be represented as binary identifiers.

```
namespace Identifier;

interface BinaryIdentifier extends Identifier
{
    /**
     * Returns the identifier as a raw byte string
     */
    public function toBytes(): string;
}
```

5.1.3 DateTimeIdentifier

A date-time identifier is based on a date-time value or otherwise has a date-time value associated with it that may be extracted from the identifier.

```
namespace Identifier;

interface DateTimeIdentifier extends Identifier
{
    /**
     * Returns a date-time representation of the timestamp associated with
     * this identifier
     */
    public function getDateTime(): \DateTimeImmutable;
}
```

5.1.4 IntegerIdentifier

Integer identifiers are identifiers that may be represented as integers.

Implementations MAY support identifiers greater than `PHP_INT_MAX` and less than `PHP_INT_MIN`. In this case, the `toInteger()` method SHOULD return a string value. If the return value is a string, it MUST be a numeric string.

If an implementation does not support identifiers greater than `PHP_INT_MAX` or less than `PHP_INT_MIN`, and the `toInteger()` operation attempts to return an integer outside these boundaries, it MUST throw an `\Identifier\Exception\OutOfRange` exception.

```php
namespace Identifier;

interface IntegerIdentifier extends Identifier
{
    /**
     * Returns an integer representation of the identifier
     *
     * @throws Exception\OutOfRange MUST throw if the implementation does
     *     not support integers outside the range of PHP_INT_MIN and
     *     PHP_INT_MAX and the identifier evaluates to an integer outside
     *     this range
     *
     * @psalm-return int | numeric-string
     */
    public function toInteger(): int | string;
}
```

5.2 Identifier Factories

5.2.1 IdentifierFactory

`IdentifierFactory` defines a common interface for factories that create identifiers.

Descendants of `IdentifierFactory` MAY specify a narrower return type for the `create()` method.

```php
namespace Identifier;

interface IdentifierFactory
{
    /**
     * Creates a new instance of an identifier
     */
    public function create(): Identifier;
}
```

5.2.2 BinaryIdentifierFactory

`BinaryIdentifierFactory` defines a common interface for factories that create identifiers from raw byte strings.

```php
namespace Identifier;

interface BinaryIdentifierFactory extends IdentifierFactory
{
    public function create(): BinaryIdentifier;

    /**
     * Creates a new instance of an identifier from the given byte string
     * representation
     *
     * @param string $identifier A binary octet string encoded according to
```

(continues on next page)

(continued from previous page)

```
     *     the specification for the type of identifier
     *
     * @throws Exception\InvalidArgument MUST throw if $identifier is not a
     *     legal value
     */
    public function createFromBytes(string $identifier): BinaryIdentifier;
}
```

5.2.3 DateTimeIdentifierFactory

`DateTimeIdentifierFactory` defines a common interface for factories that create identifiers from date-time values.

```
namespace Identifier;

interface DateTimeIdentifierFactory extends IdentifierFactory
{
    public function create(): DateTimeIdentifier;

    /**
     * Creates a new instance of an identifier from the given date-time
     *
     * @param \DateTimeInterface $dateTime The date-time to use when
     *     creating the identifier
     *
     * @throws Exception\InvalidArgument MUST throw if $dateTime is not a
     *     legal value
     */
    public function createFromDateTime(\DateTimeInterface $dateTime): DateTimeIdentifier;
}
```

5.2.4 IntegerIdentifierFactory

`IntegerIdentifierFactory` defines a common interface for factories that create identifiers from integers.

```
namespace Identifier;

interface IntegerIdentifierFactory extends IdentifierFactory
{
    public function create(): IntegerIdentifier;

    /**
     * Creates a new instance of an identifier from the given integer
     * representation
     *
     * @throws Exception\InvalidArgument MUST throw if the identifier is not
     *     a legal value
     * @throws Exception\OutOfRange MUST throw if the implementation does
     *     not support integers outside the range of PHP_INT_MIN and
     *     PHP_INT_MAX and the identifier evaluates to an integer outside
     *     this range
```

(continues on next page)

(continued from previous page)

```
     *
     * @psalm-param int | numeric-string $identifier
     */
    public function createFromInteger(int | string $identifier): IntegerIdentifier;
}
```

5.2.5 StringIdentifierFactory

`StringIdentifierFactory` defines a common interface for factories that create identifiers from strings.

```
namespace Identifier;

interface StringIdentifierFactory extends IdentifierFactory
{
    /**
     * Creates a new instance of an identifier from the given string
     * representation
     *
     * @param string $identifier The string representation of the identifier
     *     is specific to the type of identifier and implementation; for
     *     example, UUIDs use a specific format, while other identifiers may
     *     use other formats
     *
     * @throws Exception\InvalidArgument MUST throw if the identifier is not
     *     a legal value
     */
    public function createFromString(string $identifier): Identifier;
}
```

5.3 Exceptions

5.3.1 IdentifierException

This is a base exception from which all identifier exceptions descend. If an implementation uses custom exception types, they MAY implement this interface.

```
namespace Identifier\Exception;

/**
 * Base interface representing generic exceptions for identifiers
 */
interface IdentifierException extends \Throwable
{
}
```

5.3.2 InvalidArgument

Identifier factory methods that accept arguments MUST throw `InvalidArgument` exceptions if any of the arguments are not legal values for the method.

```
namespace Identifier\Exception;

/**
 * The argument passed is invalid
 */
interface InvalidArgument extends IdentifierException
{
}
```

5.3.3 NotComparable

If an implementation cannot or chooses not to compare a given value to an identifier (through `Identifier::compareTo()` or `Identifier::equals()`), it MUST throw a `NotComparable` exception.

```
namespace Identifier\Exception;

/**
 * The given value cannot be compared to the identifier
 */
interface NotComparable extends IdentifierException
{
}
```

5.3.4 OutOfRange

If an implementation does not support integer values outside the range *[PHP_INT_MIN .. PHP_INT_MAX]*, it MUST throw an `OutOfRange` exception when encountering integers outside this range as input arguments to `IntegerIdentifierFactory::createFromInteger()` or when evaluating return values for `IntegerIdentifier::toInteger()`.

```
namespace Identifier\Exception;

/**
 * The value is out of the range of acceptable values
 */
interface OutOfRange extends IdentifierException
{
}
```

CHAPTER

SIX

REFERENCES

CHAPTER

SEVEN

AUTHOR'S ADDRESSES

Ben Ramsey
Email: <ben@benramsey.com>

CHAPTER

EIGHT

PUBLIC DOMAIN DEDICATION

BIBLIOGRAPHY

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/info/rfc8141>.

[Snowflake] Twitter, "Snowflake is a network service for generating unique ID numbers at high scale with some simple guarantees.", Commit b3f6a3c, May 2014, <https://github.com/twitter-archive/snowflake/releases/tag/snowflake-2010>.

[ULID] Feerasta, A., "Universally Unique Lexicographically Sortable Identifier", Commit d0c7170, May 2019, <https://github.com/ulid/spec>.

[UUID] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[Wikipedia] Wikipedia, "Identifier", Page version ID 1122384949, November 2022, <https://en.wikipedia.org/w/index.php?title=Identifier&oldid=1122384949>.