```
import Foundation
import
UniformTypeIdentifiers.NSItemProvider_UTT
ype
import UniformTypeIdentifiers.UTAdditions
import UniformTypeIdentifiers.UTCoreTypes
import UniformTypeIdentifiers.UTDefines
import UniformTypeIdentifiers.UTTagClass
import UniformTypeIdentifiers.UTType
import _Concurrency
import _StringProcessing
import _SwiftConcurrencyShims

/**
    A type representing tag classes.

    A tag class is a "kind" of label that
describes a type in another type
    system, such as a filename extension
or MIME type. A tag is a specific
    instance of a tag class: for example,
`"txt"` is a tag, and that tag is an
    instance of the tag class
`.filenameExtension` that represents the
    same type as `UTType.plainText`.

    Older API that does not use
`UTTagClass` typically uses an untyped
`String`
    or `CFString` to refer to a tag class
as a string. To get the string
    representation of a tag class, use
its `rawValue` property.
```

```
*/
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
public struct UTTagClass :
RawRepresentable {

    /// The corresponding value of the
raw type.
    ///
    /// A new instance initialized with
`rawValue` will be equivalent to this
    /// instance. For example:
    ///
    ///     enum PaperSize: String {
    ///         case A4, A5, Letter,
Legal
    ///     }
    ///
    ///     let selectedSize =
PaperSize.Letter
    ///     print(selectedSize.rawValue)
    ///     // Prints "Letter"
    ///
    ///     print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
    ///     // Prints "true"
    public let rawValue: String

    /// Creates a new instance with the
specified raw value.
    ///
    /// If there is no value of the type
```

```
that corresponds with the specified raw
    /// value, this initializer returns
`nil`. For example:
    ///
    ///     enum PaperSize: String {
    ///         case A4, A5, Letter,
Legal
    ///     }
    ///
    ///     print(PaperSize(rawValue:
"Legal"))
    ///     // Prints
"Optional("PaperSize.Legal")"
    ///
    ///     print(PaperSize(rawValue:
"Tabloid"))
    ///     // Prints "nil"
    ///
    /// - Parameter rawValue: The raw
value to use for the new instance.
    public init(rawValue: String)

    /// The raw type that can be used to
represent all values of the conforming
    /// type.
    ///
    /// Every distinct value of the
conforming type has a corresponding
unique
    /// value of the `RawValue` type, but
there may be values of the `RawValue`
    /// type that don't have a
corresponding value of the conforming
```

```
type.
    @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
    public typealias RawValue = String
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTTagClass {

    /**
        The tag class for filename
extensions such as `"txt"`.

        The leading period character is
not part of the filename extension and
        should not be included in the
tag.

        The raw value of this tag class
is `"public.filename-extension"`.
    */
    public static var filenameExtension:
UTTagClass { get }

    /**
        The tag class for MIME types such
as `"text/plain"`.

        The raw value of this tag class
is `"public.mime-type"`.
    */
    public static var mimeType:
```

```
UTTagClass { get }
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTTagClass : Equatable,
Hashable {

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
    public static func == (lhs:
UTTagClass, rhs: UTTagClass) -> Bool

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
```

```
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTTagClass :
CustomStringConvertible,
CustomDebugStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
```

```
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// A textual representation of this
instance, suitable for debugging.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(reflecting:)`
    /// initializer. This initializer
```

```
works with any type, and uses the custom
    /// `debugDescription` property for
types that conform to
    /// `CustomDebugStringConvertible`:
    ///
    ///     struct Point:
CustomDebugStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var debugDescription:
String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(reflecting: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `debugDescription`
property.
    public var debugDescription: String {
get }
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTTagClass : Codable {

    /// Encodes this value into the given
```

```
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

/**
    A structure representing a type in a
type hierarchy.
```

```
    Types may represent files on disk,
abstract data types with no on-disk
    representation, or even entirely
unrelated hierarchical classification
    systems such as hardware.

    Older API that does not use `UTType`
typically uses an untyped `String`
    or `CFString` to refer to a type by
its identifier. To get the identifier of
    a type for use with these APIs, use
the `identifier` property of this type.

    - SeeAlso:
https://developer.apple.com/library/archi
ve/documentation/FileManagement/
Conceptual/understanding_utis/
*/
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
public struct UTType : Sendable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTType {

    /**
        Create a type given a type
identifier.

        - Parameters:
            - identifier: The type
```

```
identifier.

        - Returns: A type, or `nil` if
the type identifier is not known to the
            system.
    */
    public init?(_ identifier: String)

    /**
        Create a type given a filename
extension.

        - Parameters:
            - filenameExtension: The
filename extension for which a type is
                desired.
            - supertype: Another type
that the resulting type must conform to.
                Typically, you would pass
`.data` or `.package`.

        - Returns: A type. If no types
are known to the system with the
            specified filename extension
and conformance but the inputs were
            otherwise valid, a dynamic
type may be provided. If the inputs were
            not valid, returns `nil`.

        This method is equivalent to:

        ```
        UTType(tag: filenameExtension,
```
```

```
tagClass: .filenameExtension,
conformingTo: supertype)
        ```

        To get the type of a file on
disk, use
`URLResourceValues.contentType`.
        You should not attempt to derive
the type of a file system object based
        solely on its filename extension.
    */
    public init?(filenameExtension:
String, conformingTo supertype: UTType
= .data)

    /**
        Create a type given a MIME type.

        - Parameters:
            - mimeType: The MIME type for
which a type is desired.
            - supertype: Another type
that the resulting type must conform to.
                Typically, you would pass
`.data`.

        - Returns: A type. If no types
are known to the system with the
            specified MIME type and
conformance but the inputs were otherwise
            valid, a dynamic type may be
provided. If the inputs were not valid,
            returns `nil`.
```
```

```
        This method is equivalent to:

        ```
        UTType(tag: mimeType,
tagClass: .mimeType, conformingTo:
supertype)
        ```
    */
    public init?(mimeType: String,
conformingTo supertype: UTType = .data)

    /**
        The receiver's identifier.

        A type is _identified by_ its
Uniform Type Identifier (UTI), a
        reverse-DNS string such as
`"public.jpeg"` or `"com.adobe.pdf"`. The
        type itself _has_ a UTI, but is
not itself the UTI. This terminology is
        not consistently used across
Apple's documentation.

        Older API that does not use
`UTType` typically uses an untyped
`String`
        or `CFString` to refer to a type
by its identifier.
    */
    public var identifier: String { get }

    /**
```

```
        If available, the preferred
(first available) tag of class
        `.filenameExtension`.

        Many uses of types require the
generation of a filename (e.g. when
        saving a file to disk.) If not
`nil`, the value of this property is the
        best available filename extension
for the given type, according to its
        declaration. The value of this
property is equivalent to, but more
        efficient than:

        ```


    type.tags[.filenameExtension]?.first
        ```

    */
    public var
preferredFilenameExtension: String? { get
}


    /**
        If available, the preferred
(first available) tag of class
        `.mimeType`.

        If not `nil`, the value of this
property is the best available MIME
        type for the given type,
according to its declaration. The value
of this
```

```
        property is equivalent to, but
more efficient than:

        ```
        type.tags[.mimeType]?.first
        ```
    */
    public var preferredMIMEType: String?
{ get }

    /**
        The localized description of the
type.

        If the type does not provide a
description, the system may search its
        supertypes for one. Dynamic types
never have localized descriptions even
        if their supertypes do.
    */
    public var localizedDescription:
String? { get }

    /**
        The type's version.

        Most types do not specify a
version.
    */
    public var version: Int? { get }

    /**
        The reference URL of the type.
```

```
        A reference URL is a human-
readable document describing a type. Most
        types do not specify reference
URLs.

        - Warning: This URL is not
validated in any way by the system, nor
is
            its scheme or structure
guaranteed in any way.
    */
    public var referenceURL: URL? { get }

    /**
        Whether or not the receiver is a
dynamically generated type.

        Dynamic types are recognized by
the system, but may not be directly
        declared or claimed by an
application. They are used when a file is
        encountered whose metadata has no
corresponding type known to the
        system.

        A type cannot be both declared
_and_ dynamic. You cannot construct an
        instance of `UTType` that is
neither declared nor dynamic.
    */
    public var isDynamic: Bool { get }
```

```
    /**
        Whether or not the receiver is a
type known to the system.

        A type cannot be both declared
_and_ dynamic. You cannot construct an
        instance of `UTType` that is
neither declared nor dynamic.
    */
    public var isDeclared: Bool { get }

    /**
        Whether or not the type is in the
public domain.

        Types in the public domain have
identifiers starting with `"public."`
        and are generally defined by a
standards body or by convention. They are
        never dynamic.
    */
    public var isPublic: Bool { get }
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTType {

    /**
        Tests for a conformance
relationship between the receiver and
another
        type.
```

```
        - Parameters:
            - type: The type against
which conformance should be tested.

        - Returns: If the two types are
equal, returns `true`. If the receiver
            conforms, directly or
indirectly, to `type`, returns `true`.
            Otherwise, returns `false`.

        - SeeAlso: isSupertype(of:)
        - SeeAlso: isSubtype(of:)
    */
    public func conforms(to type: UTType)
-> Bool

    /**
        Tests if the receiver is a
supertype of another type.

        - Parameters:
            - type: The type against
which conformance should be tested.

        - Returns: If `type` conforms,
directly or indirectly, to the receiver
            and is not equal to it,
returns `true`. Otherwise, returns
`false`.

        - SeeAlso: conforms(to:)
        - SeeAlso: isSubtype(of:)
```

```
    */
    public func isSupertype(of type:
UTType) -> Bool

    /**
        Tests if the receiver is a
subtype of another type.

        - Parameters:
            - type: The type against
which conformance should be tested.

        - Returns: If the receiver
conforms, directly or indirectly, to
`type`
            and is not equal to it,
returns `true`. Otherwise, returns
`false`.

        - SeeAlso: conforms(to:)
        - SeeAlso: isSupertype(of:)
    */
    public func isSubtype(of type:
UTType) -> Bool

    /**
        The set of types to which the
receiving type conforms, directly or
        indirectly.

        If you are just interested in
checking if one type conforms to another,
        it is more efficient to use
```

```
`conforms(to:)` than this property.
    */
    public var supertypes: Set<UTType> {
get }
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTType {

    /**
        Create a type given a type tag.

        - Parameters:
            - tag: The tag, such as a
filename extension, for which a type is
                desired.
            - tagClass: The class of the
tag, such as `.filenameExtension`.
            - supertype: Another type
that the resulting type must conform to.
                If `nil`, no conformance
is required.

        - Returns: A type. If no types
are known to the system with the
            specified tag but the inputs
were otherwise valid, a dynamic type
            may be provided. If the
inputs were not valid, returns `nil`.
    */
    public init?(tag: String, tagClass:
UTTagClass, conformingTo supertype:
```

```
UTType?)

    /**
        Create an array of types given a
type tag.

        - Parameters:
            - tag: The tag, such as a
filename extension, for which a set of
                types is desired.
            - tagClass: The class of the
tag, such as `.filenameExtension`.
            - supertype: Another type
that the resulting types must
                conform to. If `nil`, no
conformance is required.

        - Returns: An array of types, or
the empty array if no such types were
            available. If no types are
known to the system with the specified
            tag but the inputs were
otherwise valid, a dynamic type may be
            provided.
    */
    public static func types(tag: String,
tagClass: UTTagClass, conformingTo
supertype: UTType?) -> [UTType]

    /**
        The tag specification dictionary
of the type.
```

```
        The system does not store tag
information for non-standard tag classes.
        It normalizes string values into
arrays containing those strings. For
        instance, a value of:

        ```
        {
            "public.mime-type": "x/y",
            "nonstandard-tag-class":
"abc",
        }
        ```

        Is normalized to:

        ```
        {
            "public.mime-type": [ "x/y" ]
        }
        ```

        If you are simply looking for the
preferred filename extension or MIME
        type of a type, it is more
efficient for you to use the
        `preferredFilenameExtension` and
`preferredMIMEType` properties
        respectively.
    */
    public var tags: [UTTagClass :
[String]] { get }
}
```

```
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTType {

    /**
        Gets an active `UTType`
corresponding to a type that is declared
as
        "exported" by the current
process.

        - Parameters:
            - identifier: The type
identifier for which a type is desired.
            - parentType: A parent type
that the resulting type is expected to
                conform to. If `nil`,
conformance to either `.data` or
`.package`
                is assumed.

        - Returns: A type.

        Use this method to get types that
are exported by your application. If
        `identifier` does not correspond
to any type known to the system, the
        result is undefined.

        You would generally use this
method by assigning its value to a
        `static let` constant in an
```

```
extension of `UTType`:

        ```
        extension UTType {
            static let myFileFormat =
UTType(exportedAs:
"com.example.myfileformat")
        }
        ```
    */
    public init(exportedAs identifier:
String, conformingTo parentType: UTType?
= nil)

    /**
        Gets an active `UTType`
corresponding to a type that is declared
as
        "imported" by the current
process.

        - Parameters:
            - identifier: The type
identifier for which a type is desired.
            - parentType: A parent type
that the resulting type is expected to
                conform to. If `nil`,
conformance to either `.data` or
`.package`
                is assumed.

        - Returns: A type whose
identifier may or may not be equal to
```

```
            `identifier`, but which is
functionally equivalent.

        Use this method to get types that
are imported by your application. If
        `identifier` does not correspond
to any type known to the system, the
        result is undefined.

        You would generally use this
method in the body of a `static` computed
        property in an extension of
`UTType` as the type can change over
time:

        ```
        extension UTType {
            static var
competitorFileFormat: UTType
{ UTType(importedAs:
"com.example.competitorfileformat") }
        }
        ```


        In the general case, this method
returns a type with the same
        identifier, but if that type has
a preferred filename extension and
        another type is the preferred
type for that extension, then that
        other type is substituted.
    */
    public init(importedAs identifier:
```

```
String, conformingTo parentType: UTType?
= nil)
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTType : Equatable, Hashable {

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
    public static func == (lhs: UTType,
rhs: UTType) -> Bool

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
```

```
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
```

```
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTType :
CustomStringConvertible,
CustomDebugStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
```

```
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// A textual representation of this
instance, suitable for debugging.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(reflecting:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `debugDescription` property for
types that conform to
    /// `CustomDebugStringConvertible`:
    ///
    ///     struct Point:
CustomDebugStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var debugDescription:
String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
```

```
    ///     let s = String(reflecting: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `debugDescription`
property.
    public var debugDescription: String {
get }
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTType : ReferenceConvertible {

    /** An alias for this value type's
equivalent reference type. */
    public typealias ReferenceType =
UTTypeReference
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTType : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
```

```
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension UTType {

    /**
        A generic base type for most
things (files, directories.)

        **UTI:** public.item
    */
```

```
    public static var item: UTType {
get }

    /**
        A base type for anything
containing user-viewable document content
        (documents, pasteboard data, and
document packages.)

        Types describing files or
packages must also conform to
`UTType.data` or
        `UTType.package` in order for the
system to bind documents to them.

        **UTI:** public.content
    */
    public static var content: UTType {
get }

    /**
        A base type for content formats
supporting mixed embedded content
        (i.e., compound documents).

        **UTI:** public.composite-content

        **conforms to:** public.content
    */
    public static var compositeContent:
UTType { get }

    /**
```

```
        A data item mountable as a volume

        **UTI:** public.disk-image
    */
    public static var diskImage: UTType {
get }

    /**
        A base type for any sort of
simple byte stream, including files and
        in-memory data.

        **UTI:** public.data

        **conforms to:** public.item
    */
    public static var data: UTType {
get }

    /**
        A file system directory (includes
packages _and_ folders.)

        **UTI:** public.directory

        **conforms to:** public.item
    */
    public static var directory: UTType {
get }

    /**
        Symbolic link and alias file
types conform to this type.
```

```
        **UTI:** com.apple.resolvable
    */
    public static var resolvable: UTType
{ get }

    /**
        A symbolic link.

        **UTI:** public.symlink

        **conforms to:** public.item,
com.apple.resolvable
    */
    public static var symbolicLink:
UTType { get }

    /**
        An executable item.

        **UTI:** public.executable

        **conforms to:** public.item
    */
    public static var executable: UTType
{ get }

    /**
        A volume mount point (resolvable,
resolves to the root directory of a
        volume.)

        **UTI:** com.apple.mount-point
```

```
        **conforms to:** public.item,
com.apple.resolvable
    */
    public static var mountPoint: UTType
{ get }

    /**
        A fully-formed alias file.

        **UTI:** com.apple.alias-file

        **conforms to:** public.data,
com.apple.resolvable
    */
    public static var aliasFile: UTType {
get }

    /**
        A URL bookmark.

        **UTI:** com.apple.bookmark

        **conforms to:** public.data,
com.apple.resolvable
    */
    public static var urlBookmarkData:
UTType { get }

    /**
        Any URL.

        **UTI:** public.url
```

```
        **conforms to:** public.data
    */
    public static var url: UTType { get }

    /**
        A URL with the scheme `"file:"`.

        **UTI:** public.file-url

        **conforms to:** public.url
    */
    public static var fileURL: UTType {
get }

    /**
        The base type for all text-
encoded data, including text with markup
        (HTML, RTF, etc.).

        **UTI:** public.text

        **conforms to:** public.data,
public.content
    */
    public static var text: UTType {
get }

    /**
        Text with no markup and an
unspecified encoding.

        **UTI:** public.plain-text
```

```
        **conforms to:** public.text
    */
    public static var plainText: UTType {
get }

    /**
        Plain text encoded as UTF-8.

        **UTI:** public.utf8-plain-text

        **conforms to:** public.plain-
text
    */
    public static var utf8PlainText:
UTType { get }

    /**
        Plain text encoded as UTF-16 with
a BOM, or if a BOM is not present,
        using "external representation"
byte order (big-endian).

        **UTI:** public.utf16-external-
plain-text

        **conforms to:** public.plain-
text
    */
    public static var
utf16ExternalPlainText: UTType { get }

    /**
```

```
        Plain text encoded as UTF-16, in
native byte order, with an optional
        BOM.

        **UTI:** public.utf16-plain-text

        **conforms to:** public.plain-
text
    */
    public static var utf16PlainText:
UTType { get }

    /**
        Text containing delimited values.

        **UTI:** public.delimited-values-
text

        **conforms to:** public.text
    */
    public static var delimitedText:
UTType { get }

    /**
        Text containing comma-separated
values (.csv).

        **UTI:** public.comma-separated-
values-text

        **conforms to:**
public.delimited-values-text
    */
```

```
    public static var commaSeparatedText:
UTType { get }

    /**
        Text containing tab-separated
values.

        **UTI:** public.tab-separated-
values-text

        **conforms to:**
public.delimited-values-text
    */
    public static var tabSeparatedText:
UTType { get }

    /**
        UTF-8 encoded text containing
tab-separated values.

        **UTI:** public.utf8-tab-
separated-values-text

        **conforms to:** public.tab-
separated-values-text, public.utf8-plain-
text
    */
    public static var
utf8TabSeparatedText: UTType { get }

    /**
        Rich Text Format data.
```

```
        **UTI:** public.rtf

        **conforms to:** public.text
    */
    public static var rtf: UTType { get }

    /**
        Any version of HTML.

        **UTI:** public.html

        **conforms to:** public.text
    */
    public static var html: UTType {
get }

    /**
        Generic XML.

        **UTI:** public.xml

        **conforms to:** public.text
    */
    public static var xml: UTType { get }

    /**
        Yet Another Markup Language.

        **UTI:** public.yaml

        **conforms to:** public.text
    */
    public static var yaml: UTType {
```

```
get }

    /**
        Cascading Style Sheets (CSS)

        **UTI:** public.css

        **conforms to:** public.text
    */
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public static var css: UTType { get }

    /**
        Abstract type for source code of
any language.

        **UTI:** public.source-code

        **conforms to:** public.plain-
text
    */
    public static var sourceCode: UTType
{ get }

    /**
        Assembly language source (.s)

        **UTI:** public.assembly-source

        **conforms to:** public.source-
code
    */
```

```
    public static var
assemblyLanguageSource: UTType { get }

    /**
        C source code (.c)

        **UTI:** public.c-source

        **conforms to:** public.source-
code
    */
    public static var cSource: UTType {
get }

    /**
        Objective-C source code (.m)

        **UTI:** public.objective-c-
source

        **conforms to:** public.source-
code
    */
    public static var objectiveCSource:
UTType { get }

    /**
        Swift source code (.swift)

        **UTI:** public.swift-source

        **conforms to:** public.source-
code
```

```
    */
    public static var swiftSource: UTType
{ get }

    /**
        C++ source code (.cp, etc.)

        **UTI:** public.c-plus-plus-
source

        **conforms to:** public.source-
code
    */
    public static var cPlusPlusSource:
UTType { get }

    /**
        Objective-C++ source code.

        **UTI:** public.objective-c-plus-
plus-source

        **conforms to:** public.source-
code
    */
    public static var
objectiveCPlusPlusSource: UTType { get }

    /**
        A C header.

        **UTI:** public.c-header
```

```
        **conforms to:** public.source-
code
    */
    public static var cHeader: UTType {
get }

    /**
        A C++ header.

        **UTI:** public.c-plus-plus-
header

        **conforms to:** public.source-
code
    */
    public static var cPlusPlusHeader:
UTType { get }

    /**
        A base type for any scripting
language source.

        **UTI:** public.script

        **conforms to:** public.source-
code
    */
    public static var script: UTType {
get }

    /**
        An AppleScript text-based script
(.applescript).
```

```
        **UTI:**
com.apple.applescript.text

        **conforms to:** public.script
    */
    public static var appleScript: UTType
{ get }

    /**
        An Open Scripting Architecture
binary script (.scpt).

        **UTI:**
com.apple.applescript.script

        **conforms to:** public.data,
public.script
    */
    public static var osaScript: UTType {
get }

    /**
        An Open Scripting Architecture
script bundle (.scptd).

        **UTI:**
com.apple.applescript.script-bundle

        **conforms to:**
com.apple.bundle, com.apple.package,
public.script
    */
```

```
    public static var osaScriptBundle:
UTType { get }

    /**
        JavaScript source code

        **UTI:** com.netscape.javascript-
source

        **conforms to:** public.source-
code, public.executable
    */
    public static var javaScript: UTType
{ get }

    /**
        The base type for shell scripts.

        **UTI:** public.shell-script

        **conforms to:** public.script
    */
    public static var shellScript: UTType
{ get }

    /**
        A Perl script.

        **UTI:** public.perl-script

        **conforms to:** public.shell-
script
    */
```

```
    public static var perlScript: UTType
{ get }

    /**
        A Python script.

        **UTI:** public.python-script

        **conforms to:** public.shell-
script
    */
    public static var pythonScript:
UTType { get }

    /**
        A Ruby script.

        **UTI:** public.ruby-script

        **conforms to:** public.shell-
script
    */
    public static var rubyScript: UTType
{ get }

    /**
        A PHP script.

        **UTI:** public.php-script

        **conforms to:** public.shell-
script
    */
```

```
    public static var phpScript: UTType {
get }

    /**
        A makefile.

        **UTI:** public.make-source

        **conforms to:** public.script
    */
    @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
    public static var makefile: UTType {
get }

    /**
        JavaScript object notation (JSON)
data

        **UTI:** public.json

        **conforms to:** public.text

        - Note: JSON almost (but doesn't
quite) conforms to
            com.netscape.javascript-
source.
    */
    public static var json: UTType {
get }

    /**
        A base type for property lists.
```

```
        **UTI:** com.apple.property-list

        **conforms to:** public.data
    */
    public static var propertyList:
UTType { get }

    /**
        An XML property list.

        **UTI:** com.apple.xml-property-
list

        **conforms to:** public.xml,
com.apple.property-list
    */
    public static var xmlPropertyList:
UTType { get }

    /**
        A binary property list.

        **UTI:** com.apple.binary-
property-list

        **conforms to:**
com.apple.property-list
    */
    public static var binaryPropertyList:
UTType { get }

    /**
```

```
        An Adobe PDF document.

        **UTI:** com.adobe.pdf

        **conforms to:** public.data,
public.composite-content
    */
    public static var pdf: UTType { get }

    /**
        A Rich Text Format Directory
document (RTF with content embedding
        in its on-disk format.)

        **UTI:** com.apple.rtfd

        **conforms to:**
com.apple.package, public.composite-
content
    */
    public static var rtfd: UTType {
get }

    /**
        A flattened RTFD document
(formatted for the pasteboard.)

        **UTI:** com.apple.flat-rtfd

        **conforms to:** public.data,
public.composite-content
    */
    public static var flatRTFD: UTType {
```

```
get }

    /**
        The WebKit webarchive format.

        **UTI:** com.apple.webarchive

        **conforms to:** public.data,
public.composite-content
    */
    public static var webArchive: UTType
{ get }

    /**
        A base type for abstract image
data.

        **UTI:** public.image

        **conforms to:** public.data,
public.content
    */
    public static var image: UTType { get
}

    /**
        A JPEG image.

        **UTI:** public.jpeg

        **conforms to:** public.image
    */
    public static var jpeg: UTType {
```

```
get }

    /**
        A TIFF image.

        **UTI:** public.tiff

        **conforms to:** public.image
    */
    public static var tiff: UTType {
get }

    /**
        A GIF image.

        **UTI:** com.compuserve.gif

        **conforms to:** public.image
    */
    public static var gif: UTType { get }

    /**
        A PNG image.

        **UTI:** public.png

        **conforms to:** public.image
    */
    public static var png: UTType { get }

    /**
        Apple icon data
```

```
        **UTI:** com.apple.icns

        **conforms to:** public.image
    */
    public static var icns: UTType {
get }

    /**
        A Windows bitmap.

        **UTI:** com.microsoft.bmp

        **conforms to:** public.image
    */
    public static var bmp: UTType { get }

    /**
        Windows icon data

        **UTI:** com.microsoft.ico

        **conforms to:** public.image
    */
    public static var ico: UTType { get }

    /**
        A base type for raw image data
(.raw).

        **UTI:** public.camera-raw-image

        **conforms to:** public.image
    */
```

```
    public static var rawImage: UTType {
get }

    /**
        A Scalable Vector Graphics image.

        **UTI:** public.svg-image

        **conforms to:** public.image
    */
    public static var svg: UTType { get }

    /**
        A Live Photo.

        **UTI:** com.apple.live-photo
    */
    public static var livePhoto: UTType {
get }

    /**
        A High Efficiency Image File
Format image.

        **UTI:** public.heif

        **conforms to:** public.heif-
standard
    */
    public static var heif: UTType {
get }

    /**
```

```
        A High Efficiency Image Coding
image.

        **UTI:** public.heic

        **conforms to:** public.heif-
standard
    */
    public static var heic: UTType {
get }

    /**
        A High Efficiency Image Coding
Image Sequence.

        **UTI:** public.heics

        **conforms to:** public.heif-
standard
    */
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public static var heics: UTType { get
}

    /**
        The WebP image format.

        **UTI:** org.webmproject.webp

        **conforms to:** public.image
    */
    public static var webP: UTType {
```

```
get }

    /**
        An EXR image.

        **UTI:** com.ilm.openexr-image

        **conforms to:** public.image
    */
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public static var exr: UTType { get }

    /**
        An Adobe DNG (digital negative)
image.

        **UTI:** com.adobe.raw-image

        **conforms to:** public.image
    */
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public static var dng: UTType { get }

    /**
        A JPEG-XL encoded image.

        **UTI:** public.jpeg-xl

        **conforms to:** public.image
    */
    @available(macOS 15.2, iOS 18.2,
```

```
watchOS 11.2, tvOS 18.2, *)
    public static var jpegxl: UTType {
get }

    /**
        A base type for 3D content.

        **UTI:** public.3d-content

        **conforms to:** public.content
    */
    public static var threeDContent:
UTType { get }

    /**
        Universal Scene Description
content.

        **UTI:** com.pixar.universal-
scene-description

        **conforms to:** public.3d-
content, public.data
    */
    public static var usd: UTType { get }

    /**
        Universal Scene Description
Package content.

        **UTI:** com.pixar.universal-
scene-description-mobile
```

```
        **conforms to:** public.3d-
content, public.data
    */
    public static var usdz: UTType {
get }

    /**
        A Reality File.

        **UTI:** com.apple.reality

        **conforms to:** public.data
    */
    public static var realityFile: UTType
{ get }

    /**
        A SceneKit serialized scene.

        **UTI:** com.apple.scenekit.scene

        **conforms to:** public.3d-
content, public.data
    */
    public static var sceneKitScene:
UTType { get }

    /**
        An AR reference object.

        **UTI:** com.apple.arobject

        **conforms to:** public.data
```

```
    */
    public static var arReferenceObject:
UTType { get }

    /**
        Any audio and/or video content.

        **UTI:** public.audiovisual-
content

        **conforms to:** public.data,
public.content
    */
    public static var audiovisualContent:
UTType { get }

    /**
        A media format which may contain
both video and audio.

        This type corresponds to what
users would label a "movie".

        **UTI:** public.movie

        **conforms to:**
public.audiovisual-content
    */
    public static var movie: UTType { get
}

    /**
        Pure video data with no audio
```

```
data.

        **UTI:** public.video

        **conforms to:** public.movie
    */
    public static var video: UTType { get
}

    /**
        Pure audio data with no video
data.

        **UTI:** public.audio

        **conforms to:**
public.audiovisual-content
    */
    public static var audio: UTType { get
}

    /**
        A QuickTime movie.

        **UTI:** com.apple.quicktime-
movie

        **conforms to:** public.movie
    */
    public static var quickTimeMovie:
UTType { get }

    /**
```

```
        An MPEG-1 or MPEG-2 movie.

        **UTI:** public.mpeg

        **conforms to:** public.movie
    */
    public static var mpeg: UTType {
get }

    /**
        An MPEG-2 video.

        **UTI:** public.mpeg-2-video

        **conforms to:** public.video
    */
    public static var mpeg2Video: UTType
{ get }

    /**
        The MPEG-2 Transport Stream movie
format.

        **UTI:** public.mpeg-2-transport-
stream

        **conforms to:** public.movie
    */
    public static var
mpeg2TransportStream: UTType { get }

    /**
        MP3 audio.
```

```
        **UTI:** public.mp3

        **conforms to:** public.audio
    */
    public static var mp3: UTType { get }

    /**
        MPEG-4 movie

        **UTI:** public.mpeg-4

        **conforms to:** public.movie
    */
    public static var mpeg4Movie: UTType
{ get }

    /**
        An MPEG-4 audio layer file.

        **UTI:** public.mpeg-4-audio

        **conforms to:** public.mpeg-4,
public.audio
    */
    public static var mpeg4Audio: UTType
{ get }

    /**
        The Apple protected MPEG4 format
(.m4p, iTunes music store format.)

        **UTI:** com.apple.protected-
```

```
mpeg-4-audio

         **conforms to:** public.audio
     */
     public static var
appleProtectedMPEG4Audio: UTType { get }

     /**
         An Apple protected MPEG-4 movie.

         **UTI:** com.apple.protected-
mpeg-4-video

         **conforms to:** com.apple.m4v-
video
     */
     public static var
appleProtectedMPEG4Video: UTType { get }

     /**
         The AVI movie format.

         **UTI:** public.avi

         **conforms to:** public.movie
     */
     public static var avi: UTType { get }

     /**
         The AIFF audio format

         **UTI:** public.aiff-audio
```

```
        **conforms to:** public.aifc-
audio
    */
    public static var aiff: UTType {
get }

    /**
        The Microsoft waveform audio
format (.wav).

        **UTI:** com.microsoft.waveform-
audio

        **conforms to:** public.audio
    */
    public static var wav: UTType { get }

    /**
        The MIDI audio format.

        **UTI:** public.midi-audio

        **conforms to:** public.audio
    */
    public static var midi: UTType {
get }

    /**
        The base type for playlists.

        **UTI:** public.playlist
    */
    public static var playlist: UTType {
```

```
get }

    /**
        An M3U or M3U8 playlist

        **UTI:** public.m3u-playlist

        **conforms to:** public.text,
public.playlist
    */
    public static var m3uPlaylist: UTType
{ get }

    /**
        A user-browsable directory (i.e.
not a package.)

        **UTI:** public.folder

        **conforms to:** public.directory
    */
    public static var folder: UTType {
get }

    /**
        The root folder of a volume or
mount point.

        **UTI:** public.volume

        **conforms to:** public.folder
    */
    public static var volume: UTType {
```

```
get }

    /**
        A packaged directory.

        Bundles differ from packages in
that a bundle has an internal file
hierarchy
        that `CFBundle` can read, while
packages are displayed to the user as if
        they were regular files. A single
file system object can be both a package
        and a bundle.

        **UTI:** com.apple.package

        **conforms to:** public.directory
    */
    public static var package: UTType {
get }

    /**
        A directory conforming to one of
the `CFBundle` layouts.

        Bundles differ from packages in
that a bundle has an internal file
hierarchy
        that `CFBundle` can read, while
packages are displayed to the user as if
        they were regular files. A single
file system object can be both a package
        and a bundle.
```

```
        **UTI:** com.apple.bundle

        **conforms to:** public.directory
    */
    public static var bundle: UTType {
get }

    /**
        The base type for bundle-based
plugins.

        **UTI:** com.apple.plugin

        **conforms to:**
com.apple.bundle, com.apple.package
    */
    public static var pluginBundle:
UTType { get }

    /**
        A Spotlight metadata importer
bundle.

        **UTI:** com.apple.metadata-
importer

        **conforms to:** com.apple.plugin
    */
    public static var spotlightImporter:
UTType { get }

    /**
```

```
        A QuickLook preview generator
bundle.

        **UTI:** com.apple.quicklook-
generator

        **conforms to:** com.apple.plugin
    */
    public static var quickLookGenerator:
UTType { get }

    /**
        An XPC service bundle.

        **UTI:** com.apple.xpc-service

        **conforms to:**
com.apple.bundle, com.apple.package
    */
    public static var xpcService: UTType
{ get }

    /**
        A macOS or iOS framework bundle.

        **UTI:** com.apple.framework

        **conforms to:** com.apple.bundle
    */
    public static var framework: UTType {
get }

    /**
```

```
        The base type for macOS and iOS
applications.

        **UTI:** com.apple.application

        **conforms to:**
public.executable
    */
    public static var application: UTType
{ get }

    /**
        A bundled application.

        **UTI:** com.apple.application-
bundle

        **conforms to:**
com.apple.application, com.apple.bundle,
com.apple.package
    */
    public static var applicationBundle:
UTType { get }

    /**
        An application extension
(.appex).

        **UTI:** com.apple.application-
and-system-extension

        **conforms to:** com.apple.xpc-
service
```

```
    */
    public static var
applicationExtension: UTType { get }

    /**
        A UNIX executable (flat file.)

        **UTI:** public.unix-executable

        **conforms to:** public.data,
public.executable
    */
    public static var unixExecutable:
UTType { get }

    /**
        A Windows executable (.exe).

        **UTI:** com.microsoft.windows-
executable

        **conforms to:** public.data,
public.executable
    */
    public static var exe: UTType { get }

    /**
        A System Preferences pane.

        **UTI:**
com.apple.systempreference.prefpane

        **conforms to:**
```

```
com.apple.package, com.apple.bundle
    */
    public static var
systemPreferencesPane: UTType { get }

    /**
        an archive of files and
directories

        **UTI:** public.archive
    */
    public static var archive: UTType {
get }

    /**
        A GNU zip archive.

        **UTI:** org.gnu.gnu-zip-archive

        **conforms to:** public.data,
public.archive
    */
    public static var gzip: UTType {
get }

    /**
        A bzip2 archive.

        **UTI:** public.bzip2-archive

        **conforms to:** public.data,
public.archive
    */
```

```
    public static var bz2: UTType { get }

    /**
        A zip archive.

        **UTI:** public.zip-archive

        **conforms to:** com.pkware.zip-
archive
    */
    public static var zip: UTType { get }

    /**
        An Apple Archive.

        **UTI:** com.apple.archive

        **conforms to:** public.data,
public.archive
    */
    public static var appleArchive:
UTType { get }

    /**
        A tar Archive.

        **UTI:** public.tar-archive

        **conforms to:** public.data,
public.archive
    */
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
```

```
    public static var tarArchive: UTType
{ get }

    /**
        A base type for spreadsheet
documents.

        **UTI:** public.spreadsheet

        **conforms to:** public.content
    */
    public static var spreadsheet: UTType
{ get }

    /**
        A base type for presentation
documents.

        **UTI:** public.presentation

        **conforms to:**
public.composite-content
    */
    public static var presentation:
UTType { get }

    /**
        A database store.

        **UTI:** public.database
    */
    public static var database: UTType {
get }
```

```
    /**
        A base type for messages (email,
IM, etc.)

        **UTI:** public.message
    */
    public static var message: UTType {
get }

    /**
        contact information, e.g. for a
person, group, organization

        **UTI:** public.contact
    */
    public static var contact: UTType {
get }

    /**
        A vCard file.

        **UTI:** public.vcard

        **conforms to:** public.text,
public.contact
    */
    public static var vCard: UTType { get
}

    /**
        A to-do item.
```

```
        **UTI:** public.to-do-item
    */
    public static var toDoItem: UTType {
get }

    /**
        A calendar event.

        **UTI:** public.calendar-event
    */
    public static var calendarEvent:
UTType { get }

    /**
        An e-mail message.

        **UTI:** public.email-message

        **conforms to:** public.message
    */
    public static var emailMessage:
UTType { get }

    /**
        A base type for Apple Internet
location files.

        **UTI:** com.apple.internet-
location

        **conforms to:** public.data
    */
    public static var internetLocation:
```

```
UTType { get }

    /**
        Microsoft Internet shortcut files
(.url).

        **UTI:** com.apple.internet-
location

        **conforms to:** public.data
    */
    public static var internetShortcut:
UTType { get }

    /**
        A base type for fonts.

        **UTI:** public.font
    */
    public static var font: UTType {
get }

    /**
        A bookmark.

        **UTI:** public.bookmark

        - SeeAlso: UTType.urlBookmarkData
    */
    public static var bookmark: UTType {
get }

    /**
```

```
        PKCS#12 data.

        **UTI:** com.rsa.pkcs-12

        **conforms to:** public.data
    */
    public static var pkcs12: UTType {
get }

    /**
        An X.509 certificate.

        **UTI:** public.x509-certificate

        **conforms to:** public.data
    */
    public static var x509Certificate:
UTType { get }

    /**
        The EPUB format.

        **UTI:** org.idpf.epub-container

        **conforms to:** public.data,
public.composite-content
    */
    public static var epub: UTType {
get }

    /**
        A base type for console logs.
```

```
        **UTI:** public.log
    */
    public static var log: UTType { get }

    /**
        An Apple Haptics Audio Pattern
file.

        **UTI:** com.apple.haptics.ahap
    */
    @available(macOS 14.0, iOS 17.0,
watchOS 10.0, tvOS 17.0, *)
    public static var ahap: UTType {
get }

    /**
        A GeoJSON file.

        **UTI:** public.geojson
    */
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public static var geoJSON: UTType {
get }

    /**
        Serialized LinkPresentation
metadata.

        **UTI:**
com.apple.linkpresentation.metadata

        **conforms to:** public.data
```

```
    */
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public static var
linkPresentationMetadata: UTType { get }
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension URLResourceValues {

    /** The file type of the resource. */
    public var contentType: UTType? { get
}
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension URL {

    /**
        Generate a path component based
on a partial filename and a file type,
        then append it to a copy of the
receiver.

        - Parameters:
            - partialName: The partial
filename that should be expanded upon,
                e.g. `"readme"`.
            - contentType: The type the
resulting file should conform to, e.g.
                `.plainText`.
```

- Returns: A complete URL. Using the argument examples above, this
            method would return a URL with a last path component of
            `"readme.txt"`.

        Use this method when you have partial input from a user or other source
        and need to produce a complete filename suitable for that input. For
        example, if you are downloading a file from the Internet and know its
        MIME type, you can use this method to ensure the correct filename
        extension is applied to the URL where you save the file.

        If `partialName` already has a path extension, and that path extension
        is valid for file system objects of type `contentType`, no additional
        extension is appended to the path component before constructing the URL.
        For example, if the inputs are `"puppy.jpg"` and `.image` respectively,
        then the resulting URL will have a last path component of `"puppy.jpg"`.
        On the other hand, if the inputs are `"puppy.jpg"` and `.plainText`
        respectively, the resulting URL will have a last path component of
        `"puppy.jpg.txt"`. If you want to

```
ensure any existing path extension is
        replaced, you can use the
`deletingPathExtension()` method first.

        If the path component could not
be appended, this method returns `self`.

        - Note: The resulting URL has a
directory path if `contentType` conforms
            to `.directory`.
    */
    public func appendingPathComponent(_
partialName: String, conformingTo
contentType: UTType) -> URL

    /**
        Generate a path component based
on a partial filename and a file type,
        then append it to the receiver.

        - Parameters:
            - partialName: The partial
filename that should be expanded upon,
                e.g. `"readme"`.
            - contentType: The type the
resulting file should conform to, e.g.
                `.plainText`.

        Use this method when you have
partial input from a user or other source
        and need to produce a complete
filename suitable for that input. For
        example, if you are downloading a
```

```
file from the Internet and know its
        MIME type, you can use this
method to ensure the correct filename
        extension is applied to the URL
where you save the file.

        If `partialName` already has a
path extension, and that path extension
        is valid for file system objects
of type `contentType`, no additional
        extension is appended to the path
component before appending it to the
        URL. For example, if the inputs
are `"puppy.jpg"` and `.image`
        respectively, then the URL will
have a path component of `"puppy.jpg"`
        appended to it. On the other
hand, if the inputs are `"puppy.jpg"` and
        `.plainText` respectively, the
URL will have a path component of
        `"puppy.jpg.txt"` appended to it.
If you want to ensure any existing
        path extension is replaced, you
can use the `deletePathExtension()`
        method first.

        If the path component could not
be appended, `self` is not modified.

        - Note: The modified URL has a
directory path if `contentType` conforms
            to `.directory`.
    */
```

```
    public mutating func
appendPathComponent(_ partialName:
String, conformingTo contentType: UTType)

    /**
        Generate a path component based
on the last path component of the
        receiver and a file type, then
replace the last path component of a copy
        of the receiver with it.

        - Parameters:
            - contentType: The type the
resulting file should conform to, e.g.
                `.plainText`.

        - Returns: A complete URL. Using
the argument examples above, this
            method would return a URL
with a last path component of
            `"readme.txt"`.

        Use this method when you have
partial input from a user or other source
        and need to produce a complete
filename suitable for that input. For
        example, if you are downloading a
file from the Internet and know its
        MIME type, you can use this
method to ensure the correct filename
        extension is applied to the URL
where you save the file.
```

```
        If the receiver already has a
path extension, and that path extension
        is valid for file system objects
of type `contentType`, no additional
        extension is appended to URL. For
example, if the inputs are
        `"puppy.jpg"` and `.image`
respectively, then the resulting URL's
last
        path component will equal
`"puppy.jpg"`. On the other hand, if the
        inputs are `"puppy.jpg"` and
`.plainText` respectively, the resulting
        URL's last path component will
equal `"puppy.jpg.txt"`. If you want to
        ensure any existing path
extension is replaced, you can use the
        `deletePathExtension()` method
first.

        If the path component could not
be appended, this method returns `self`.

        - Note: The resulting URL has a
directory path if `contentType` conforms
            to `.directory`.
    */
    public func
appendingPathExtension(for contentType:
UTType) -> URL

    /**
        Generate a path component based
```

```
on the last path component of the
        receiver and a file type, then
replace the receiver's last path
        component with it.

        - Parameters:
            - contentType: The type the
resulting file should conform to, e.g.
                `.plainText`.

        Use this method when you have
partial input from a user or other source
        and need to produce a complete
filename suitable for that input. For
        example, if you are downloading a
file from the Internet and know its
        MIME type, you can use this
method to ensure the correct filename
        extension is applied to the URL
where you save the file.

        If the receiver already has a
path extension, and that path extension
        is valid for file system objects
of type `contentType`, no additional
        extension is appended to URL. For
example, if the inputs are
        `"puppy.jpg"` and `.image`
respectively, then the resulting URL's
last
        path component will equal
`"puppy.jpg"`. On the other hand, if the
        inputs are `"puppy.jpg"` and
```

```
`.plainText` respectively, the resulting
        URL's last path component will
equal `"puppy.jpg.txt"`. If you want to
        ensure any existing path
extension is replaced, you can use the
        `deletePathExtension()` method
first.

        If the path component could not
be appended, `self` is not modified.

        - Note: The modified URL has a
directory path if `contentType` conforms
            to `.directory`.
    */
    public mutating func
appendPathExtension(for contentType:
UTType)
}

@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension NSItemProvider {

    /// Initialize this instance with the
contents of a URL
    ///
    /// The filename of the URL is copied
into the `suggestedName` property.
    ///
    /// - Parameters:
    ///   - fileURL: The URL of the file
    ///   - contentType: The content type
```

```
associated with this file, or \c nil to
deduce the content type from the
    ///                     file extension
    ///   - openInPlace: Pass `true` to
allow this file to be opened in place, or
`false` to have this file copied.
    ///   - coordinated: Pass `true` to
use file coordination to access this
file, even if it is not opened in place.
    ///                     If `openInPlace`
is set to `true`, file coordination will
be used and this parameter is ignored.
    public convenience init(contentsOf
fileURL: URL, contentType: UTType?,
openInPlace: Bool = false, coordinated:
Bool = false, visibility:
NSItemProviderRepresentationVisibility
= .all)

    /// Register a representation backed
by `Data`
    ///
    /// The load handler must call the
completion block when loading is
complete. Pass either a non-nil `Data`,
    /// or a non-nil error. If the load
handler returns a non-nil `Progress`, it
should report loading progress
    /// and respond to cancelation.
    ///
    /// - Parameters:
    ///   - contentType: The content type
associated with the data representation.
```

```
    ///   - visibility: Specifies which
processes have access to this
representation.
    ///   - loadHandler: A block called
to provide the data representation.
    public func
registerDataRepresentation(for
contentType: UTType, visibility:
NSItemProviderRepresentationVisibility
= .all, loadHandler: @escaping @Sendable
(@escaping (Data?, (any Error)?) -> Void)
-> Progress?)

    /// Register a representation backed
by a file
    ///
    /// It is permissible to provide a
URL pointing to a folder. A folder
requested as `Data` will yield a `Data`
    /// containing a zip archive holding
a copy of the source folder tree.
    ///
    /// The load handler must call the
completion block when loading is
complete. Pass either a non-nil url,
    /// or a non-nil error. Pass `true`
to `coordinated` if the file should be
accessed using file coordination even if
    /// it is not opened in-place. Files
registered as open-in-place are assumed
to need coordination, and this
    /// parameter will be ignored in
those cases. If the load handler returns
```

```
a non-nil progress object, it should
    /// report loading progress and
respond to cancelation.
    ///
    /// - Note: Not all files specified
as openable in place can be opened in
place by the destination.
    ///         System security or
privacy policies may restrict which files
can be opened in place.
    ///
    /// - Parameters:
    ///   - contentType: The content type
associated with the file representation.
    ///   - visibility: Specifies which
processes have access to this
representation.
    ///   - openInPlace: Specifies
whether the file should be openable in
place.
    ///   - loadHandler: A block called
to provide the file representation.
    public func
registerFileRepresentation(for
contentType: UTType, visibility:
NSItemProviderRepresentationVisibility
= .all, openInPlace: Bool = false,
loadHandler: @escaping @Sendable
(@escaping (URL?, Bool, (any Error)?) ->
Void) -> Progress?)

    /// Load a representation as data
    ///
```

```
    /// If the requested representation
was registered as a file, a `Data` with
the contents of the file
    /// will be provided. If the
registered URL points to a folder, a
`Data` containing a zip archive
containing that
    /// folder will be provided.
    ///
    /// - Note: The completion handler
may be scheduled on an arbitrary queue.
    ///
    /// - Parameters:
    ///   - contentType: Content type of
the representation to load. Must conform
to one of the content types returned
    ///                  by
`registeredContentTypes`.
    ///   - completionHandler: A block
that will be called when loading is
complete. It will either have a non-nil
    ///                       `Data` or
a non-nil error.
    ///
    /// - Returns: A progress object. Use
it to monitor loading progress, or to
cancel loading.
    public func
loadDataRepresentation(for contentType:
UTType, completionHandler: @escaping
@Sendable (Data?, (any Error)?) -> Void)
-> Progress
```

```
    /// Load a representation as a file
    ///
    /// Except for files registered as
open-in-place, a temporary file
containing a copy of the original will be
    /// provided to your completion
handler. This temporary file will be
deleted once your completion handler
    /// returns. To keep a copy of this
file, move or copy it into another
directory before returning from the
    /// completion handler.
    ///
    /// If the representation was
registered as `Data`, its contents will
be written to a temporary file.
    ///
    /// If `suggestedName` is non-nil, an
attempt will be made to use it as the
file name, with an appropriate
    /// file extension based on the
content type. Otherwise, a suitable name
and file extension will be chosen based
on
    /// the content type.
    ///
    /// - Note: The completion handler
may be scheduled on an arbitrary queue.
    ///
    /// - Parameters:
    ///   - contentType: Content type of
the representation to load. Must conform
to one of the content types returned
```

```
    ///                     by
`registeredContentTypes`.
    ///   - openInPlace: Pass `true` to
attempt to open a file representation in
place.
    ///   - completionHandler: A block
that will be called when loading is
complete. It will either have a non-nil
    ///                           `URL` or a
non-nill error. The `openInPlace`
parameter will be set to
    ///                           `true` if
the file was successfully opened in
place, or `false if a copy of the file
was
    ///                           created in
a temporary directory.
    ///
    /// - Returns: A progress object. Use
it to monitor loading progress, or to
cancel loading.
    public func
loadFileRepresentation(for contentType:
UTType, openInPlace: Bool = false,
completionHandler: @escaping @Sendable
(URL?, Bool, (any Error)?) -> Void) ->
Progress
}
```