Scalars & Enumerations

The scalar and enumeration represent two of the basic types in GraphQL, together referred to as "leaf types". Both must be bound to C# types; be that an enum (as is the case with enumeration types) or to a specific value/reference type.

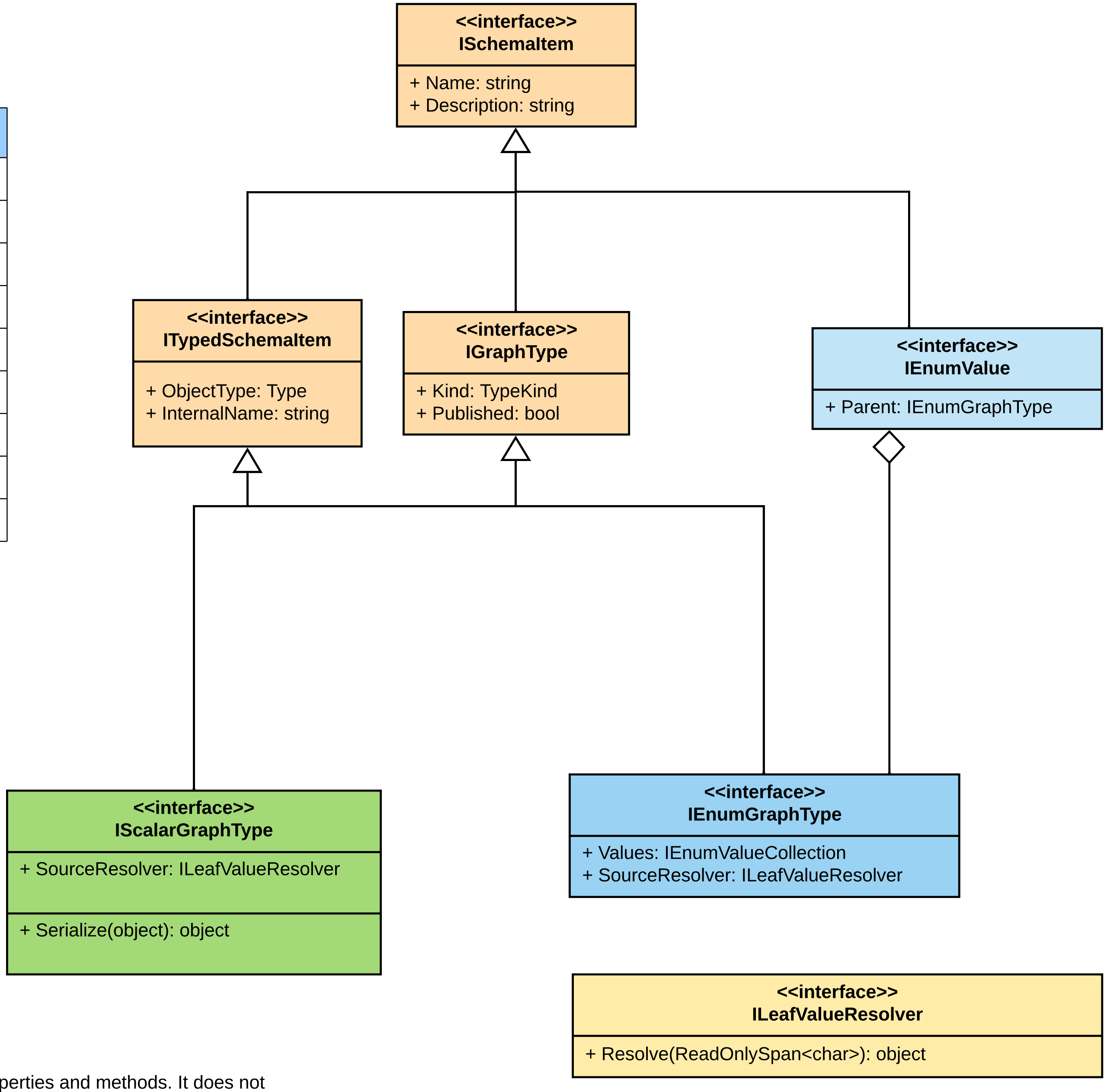
You can add custom scalars to the system by registering your own IScalarGraphType at start up.

Graph Name	.NET Type	Serialized Type
STRING	string	string
INT	int	number
UINT	uint	number
LONG	long	number
ULONG	ulong	number
DECIMAL	decimal	number
FLOAT	float	number
DOUBLE	double	number
DATE*	DateTime	number

A complete list of scalar types is available at:
https://graphql-aspnet.github.io/docs/types/scalars

** GraphQL, by default, serializes dates to a number of ticks, **in milliseconds,** from the unix epoch.*

All value type scalars can be nullable (e.g. int?) . The object graph you construct will be automatically configured for nullable values depending on the properties and methods found in your code.



<<interface>>
ISchemaItem
+ Name: string
+ Description: string

<<interface>>
ITypedSchemaItem
+ ObjectType: Type
+ InternalName: string

<<interface>>
IGraphType
+ Kind: TypeKind
+ Published: bool

<<interface>>
IEnumValue
+ Parent: IEnumGraphType

<<interface>>
IScalarGraphType
+ SourceResolver: ILeafValueResolver
+ Serialize(object): object

<<interface>>
IEnumGraphType
+ Values: IEnumValueCollection
+ SourceResolver: ILeafValueResolver

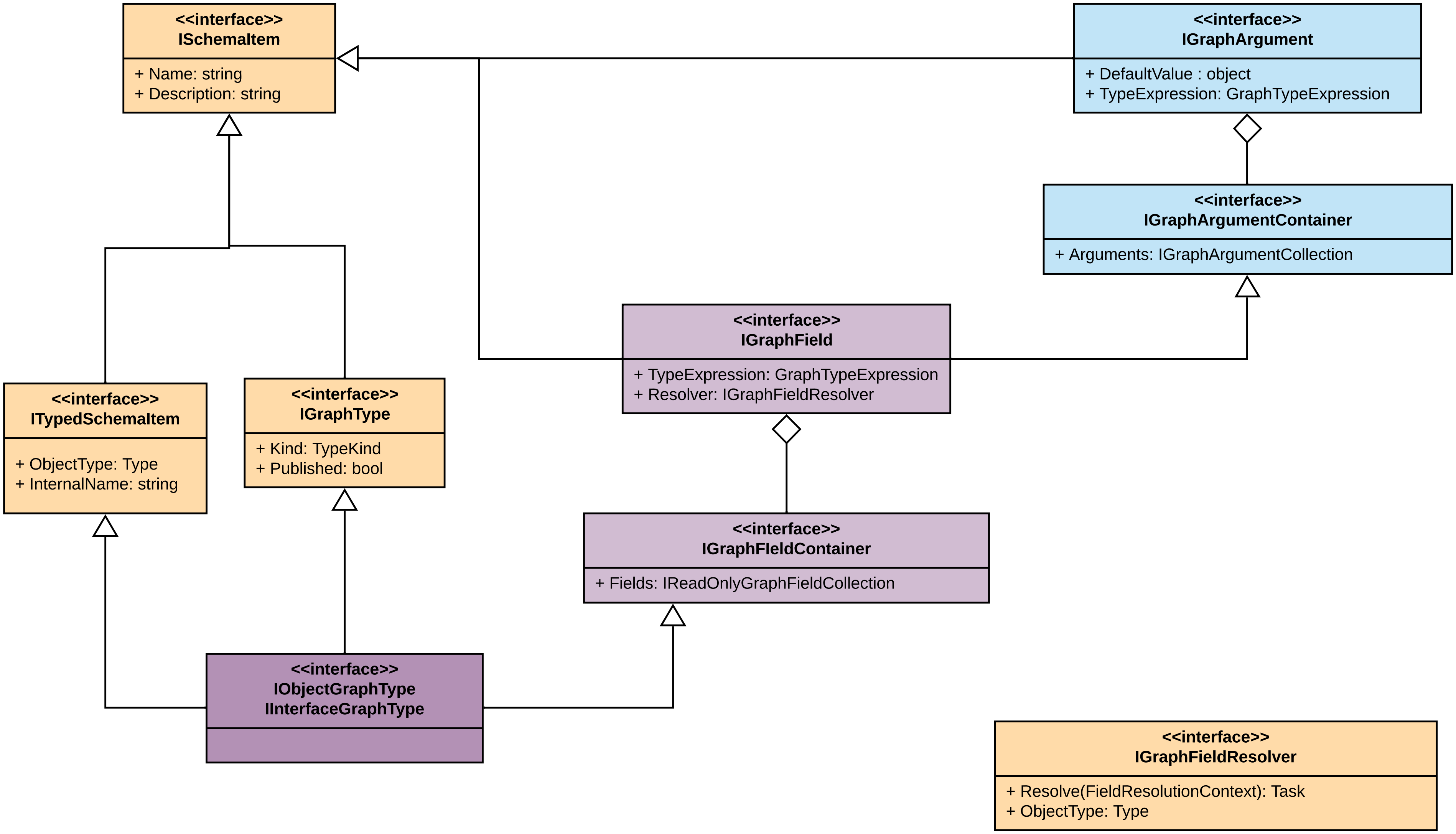
<<interface>>
ILeafValueResolver
+ Resolve(ReadOnlySpan<char>): object

* The items in this document represent the primary set of interfaces, properties and methods. It does not represent a complete model.

All represented interfaces located at: */src/graphql-aspnet/Interfaces/**

Object and Interface Graph Types

The structure of the OBJECT and INTERFACE graph types are nearly identical. They both contain field definitions that potentially have arguments. The contents of and use of these fields will vary significantly at runtime depending on the graph type in question.

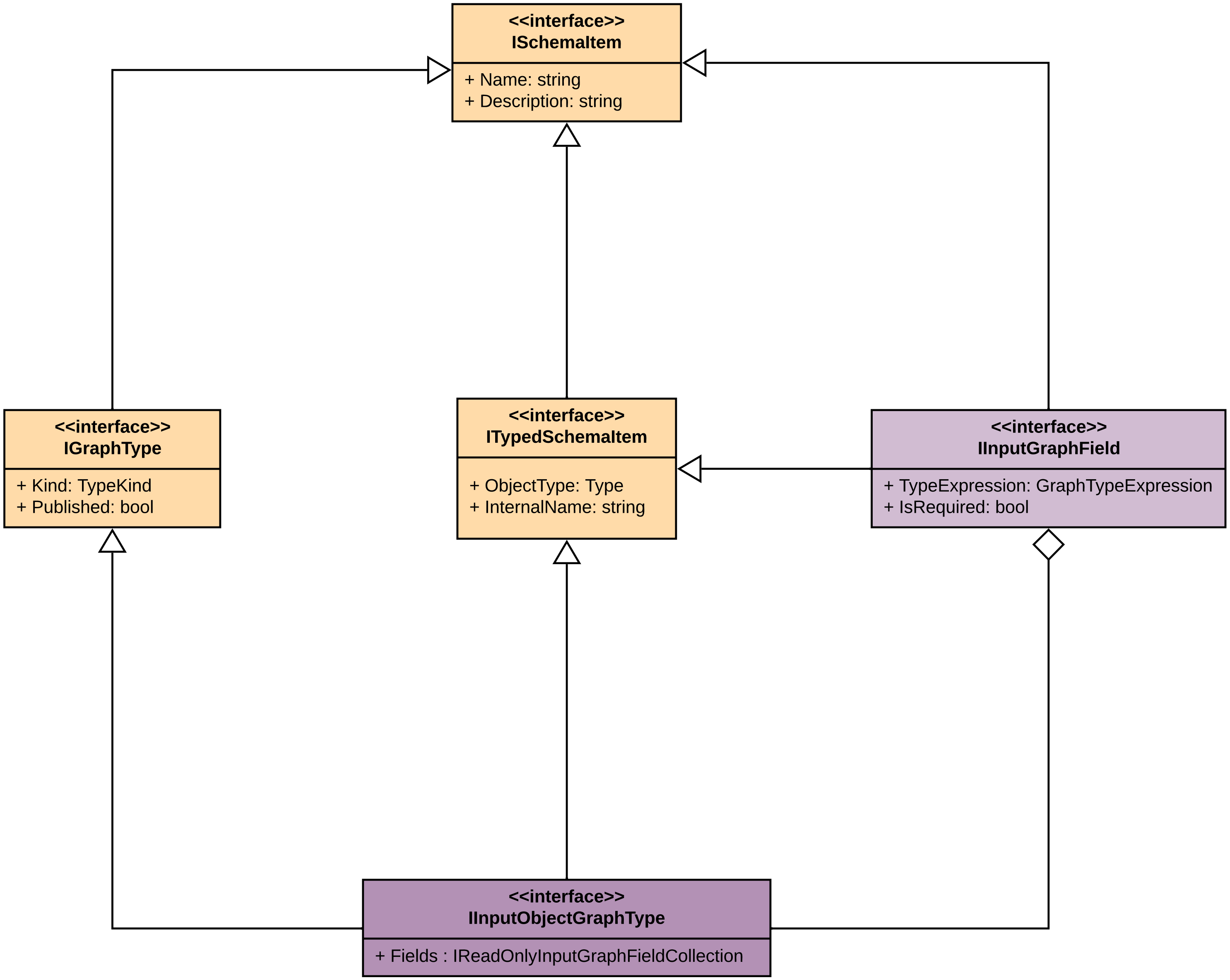


* The items in this document represent the primary set of interfaces, properties and methods. It does not represent a complete model.

All represented interfaces located at: */src/graphql-aspnet/Interfaces/**

Input Object Graph Type

The INPUT_OBJECT graph type represents complex input values (such as objects) to graph fields. For the most part it is a collection of named fields each of which may hold a leaf value or another INPUT_OBJECT.

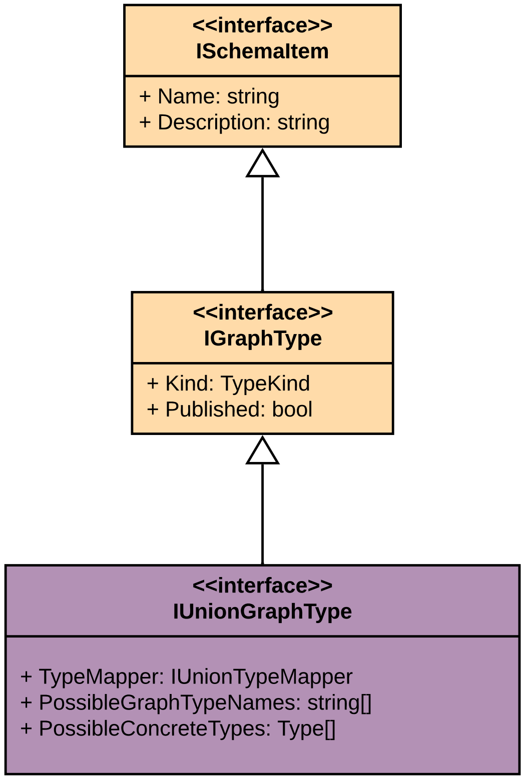


* The items in this document represent the primary set of interfaces, properties and methods. It does not represent a complete model.

All represented interfaces located at: */src/graphql-aspnet/Interfaces/**

Union Graph Type

The union type represents an intersection of other graph types. It contains a type mapper to distinguish which graph type a resolved .NET object should masquerade as when executing a query.



<<interface>>
ISchemaItem

+ Name: string
+ Description: string

<<interface>>
IGraphType

+ Kind: TypeKind
+ Published: bool

<<interface>>
IUnionGraphType

+ TypeMapper: IUnionTypeMapper
+ PossibleGraphTypeNames: string[]
+ PossibleConcreteTypes: Type[]

The union graph type represents multiple different possible graph types. It contains the names of the graph types contained in the union.

The **TypeMapper** property points to a class that can map between union types. This is used to resolve some edge cases caused by object inheritance chains when a resolved object could represent more than one type in the union.

For instance if a field resolver returned a Teacher object and the union represents both Teachers and Employee objects. Since all teachers are also employees it cannot deteremine which type is being requested without additional criteria.

<<interface>>
IUnionTypeMapper

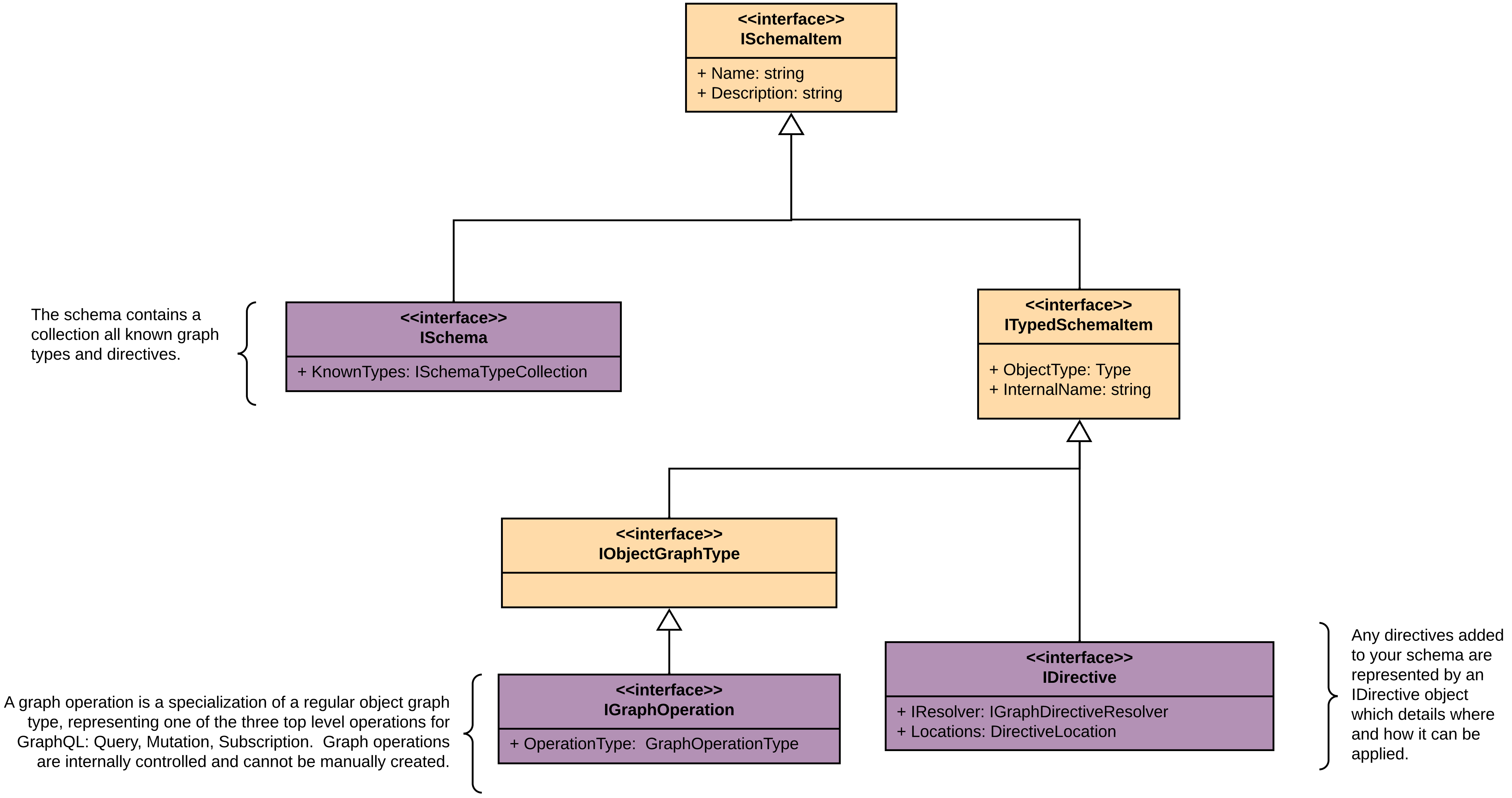
+ MapType(Type runtimeObjectType): Type

* The items in this document represent the primary set of interfaces, properties and methods. It does not represent a complete model.

All represented interfaces located at: */src/graphql-aspnet/Interfaces/**

Other Schema Items

This diagram shows some other import schema items, not related to graph types or fields.



The schema contains a collection all known graph types and directives.

A graph operation is a specialization of a regular object graph type, representing one of the three top level operations for GraphQL: Query, Mutation, Subscription. Graph operations are internally controlled and cannot be manually created.

Any directives added to your schema are represented by an IDirective object which details where and how it can be applied.

* The items in this document represent the primary set of interfaces, properties and methods. It does not represent a complete model.

All represented interfaces located at: */src/graphql-aspnet/Interfaces/**

Subscription Interfaces

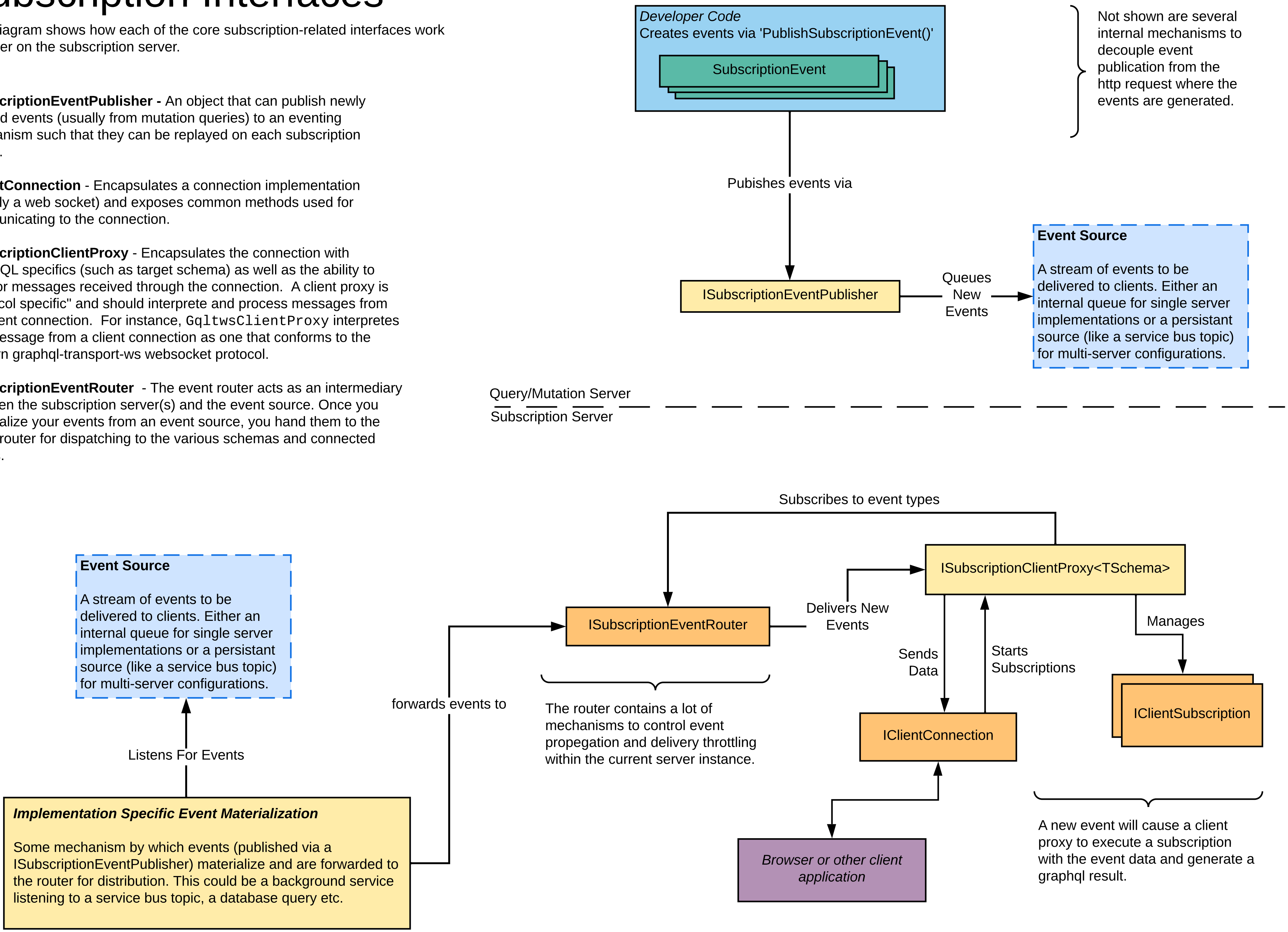
This diagram shows how each of the core subscription-related interfaces work together on the subscription server.

ISubscriptionEventPublisher - An object that can publish newly created events (usually from mutation queries) to an eventing mechanism such that they can be replayed on each subscription server.

IClientConnection - Encapsulates a connection implementation (usually a web socket) and exposes common methods used for communicating to the connection.

ISubscriptionClientProxy - Encapsulates the connection with GraphQL specifics (such as target schema) as well as the ability to monitor messages received through the connection. A client proxy is "protocol specific" and should interprete and process messages from the client connection. For instance, `GqltwsClientProxy` interpretes any message from a client connection as one that conforms to the modern graphql-transport-ws websocket protocol.

ISubscriptionEventRouter - The event router acts as an intermediary between the subscription server(s) and the event source. Once you deserialize your events from an event source, you hand them to the event router for dispatching to the various schemas and connected clients.



Developer Code
Creates events via 'PublishSubscriptionEvent()'

SubscriptionEvent

Not shown are several internal mechanisms to decouple event publication from the http request where the events are generated.

Pubishes events via

ISubscriptionEventPublisher

Queues New Events

Event Source

A stream of events to be delivered to clients. Either an internal queue for single server implementations or a persistant source (like a service bus topic) for multi-server configurations.

Query/Mutation Server

Subscription Server

Subscribes to event types

ISubscriptionClientProxy<TSchema>

Event Source

A stream of events to be delivered to clients. Either an internal queue for single server implementations or a persistant source (like a service bus topic) for multi-server configurations.

ISubscriptionEventRouter

Delivers New Events

Sends Data

Starts Subscriptions

Manages

IClientSubscription

forwards events to

The router contains a lot of mechanisms to control event propegation and delivery throttling within the current server instance.

IClientConnection

Listens For Events

Implementation Specific Event Materialization

Some mechanism by which events (published via a ISubscriptionEventPublisher) materialize and are forwarded to the router for distribution. This could be a background service listening to a service bus topic, a database query etc.

Browser or other client application

A new event will cause a client proxy to execute a subscription with the event data and generate a graphql result.

This document represents the major interfaces needed to understand how the standard subscription flow works. It is not meant to be an exaustive study of all the moving parts. Inspect the subscription library's source code for all the details.