Scalars & Enumerations

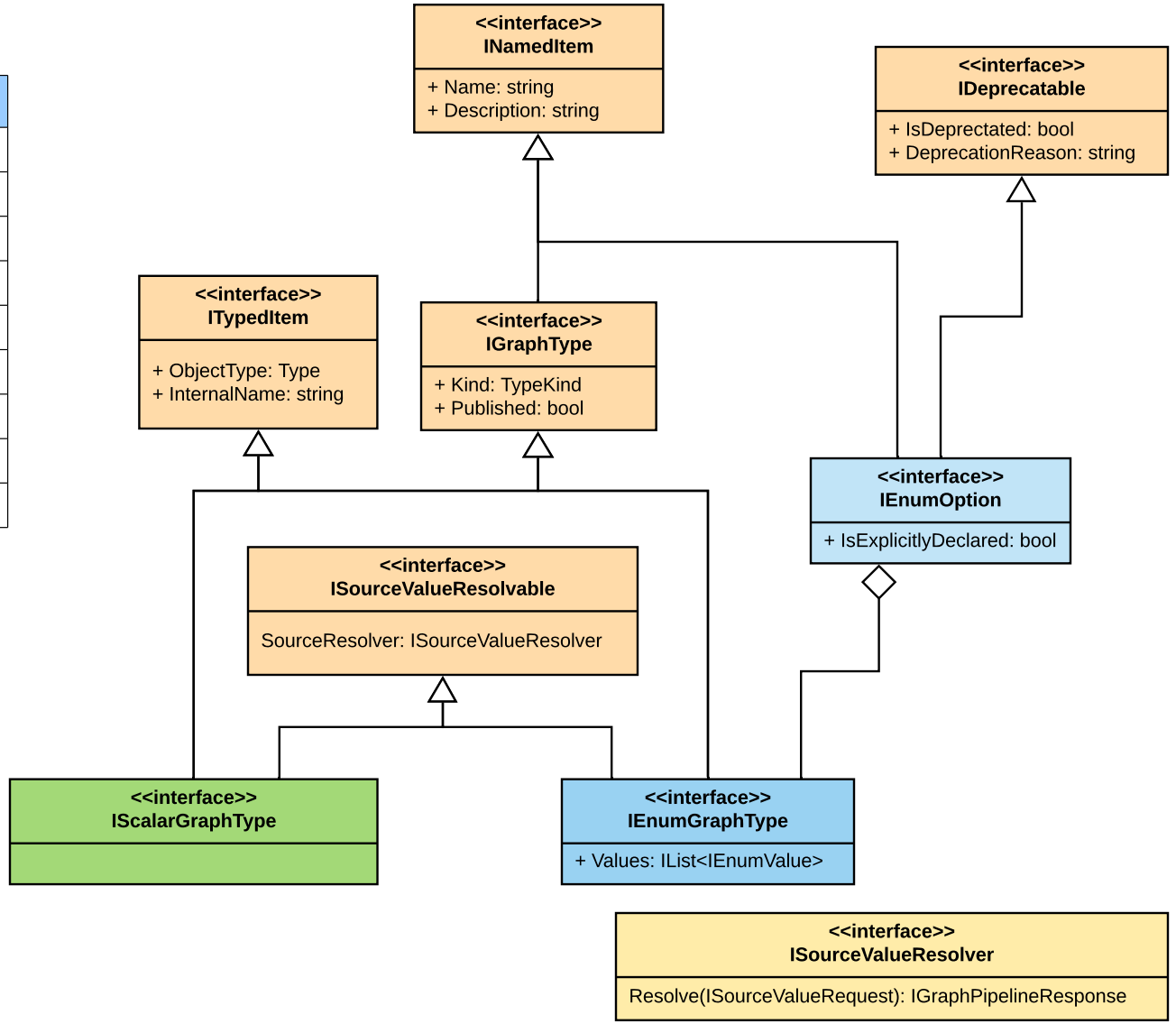
The scalar and enumeration represent two of the basic types in GraphQL, together referred to as "leaf types". Both types must be bound to C# objects; be that an enum (as is the case with enumeration types) or to a built in value/reference type in the base class library.

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

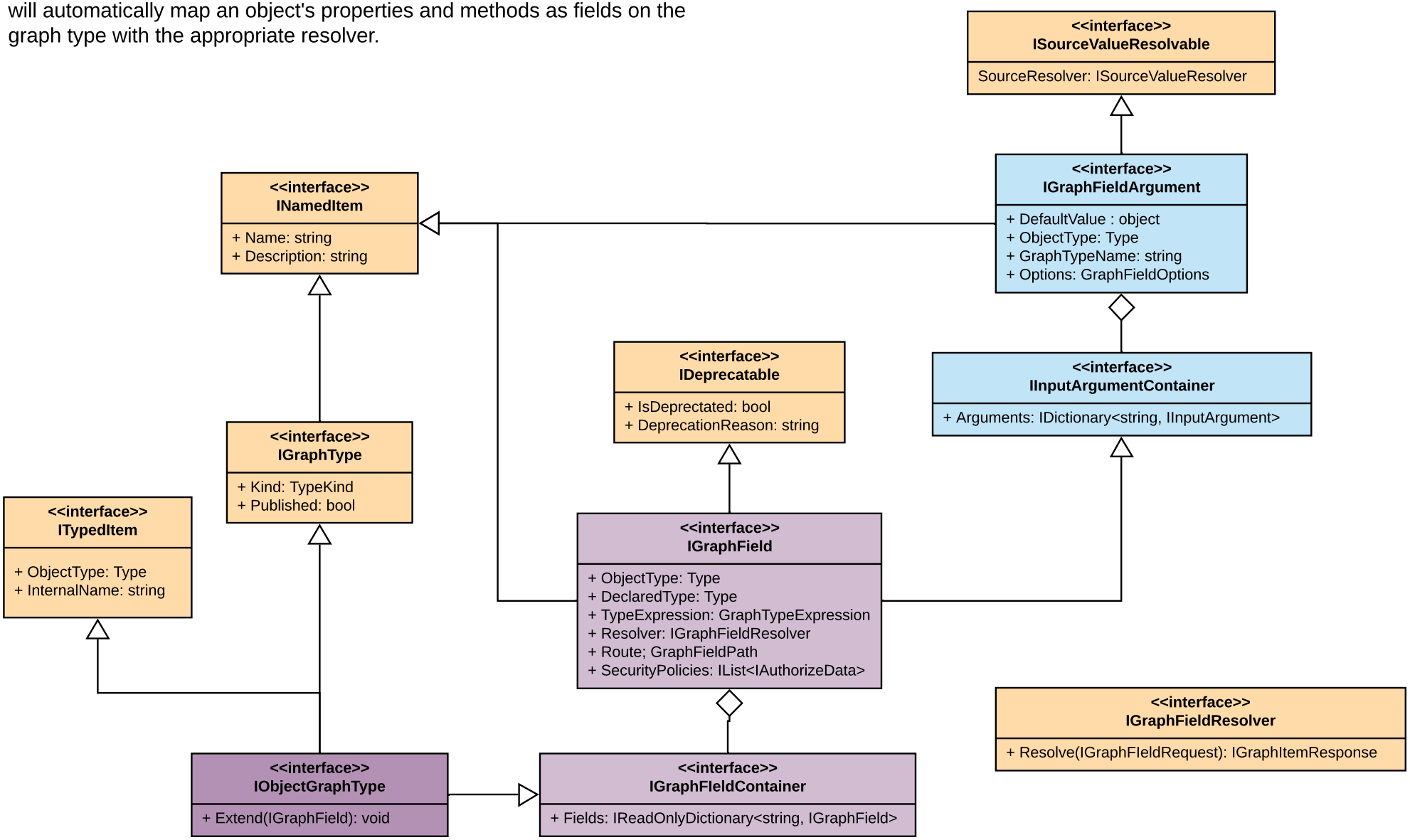
All value type scalars can be nullable (e.g. int?) . The object graph you construct will be automatically configured for nullable values depending on the property and method return types in your C# code.



The interfaces in this document do not represent a complete list of properties and methods.
All interfaces located at: */src/graphql-aspnet/Interfaces/**

Object Graph Type

The object graph type is how developer created objects are expressed and made available in the graph structure. Each concrete C# class can be mapped into one (and only one) object graph type. The templating system will automatically map an object's properties and methods as fields on the graph type with the appropriate resolver.



* The interfaces in this document do not represent a complete list of properties and methods.

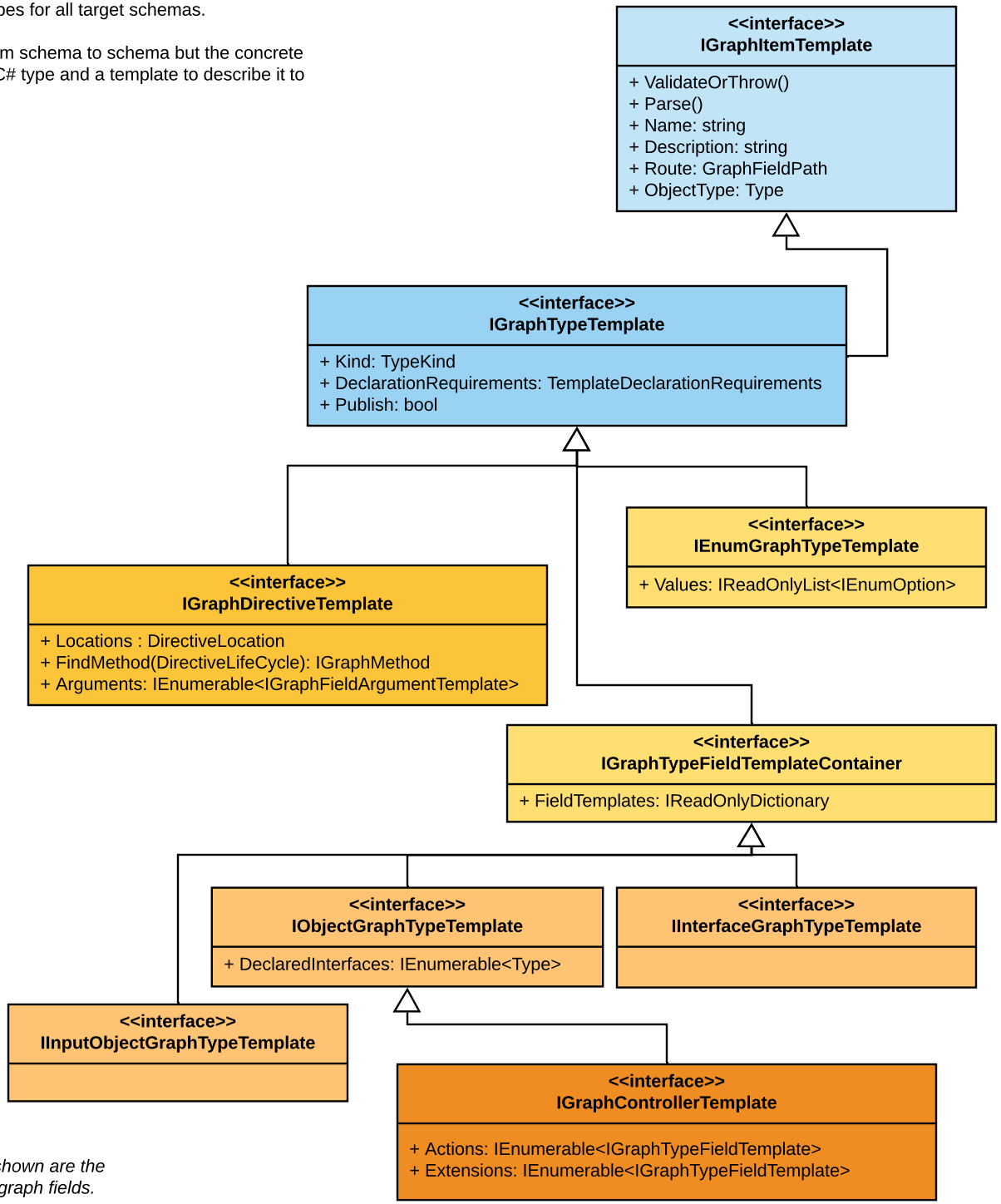
All interfaces located at: */src/graphql-aspnet/Interfaces/**

Graph Type Templates

A template harnesses a C# class and provides all the logic and data to make a graph type for a schema. Any class or enum created by a developer will have a single template coorisponding to generate the graph types for all target schemas.

Templates are universal. A graph type may vary in its name or in field composition etc. from schema to schema but the concrete type's template will be the same. In general, there is a 1:1 mapping between any defined C# type and a template to describe it to the runtime.

Template	Purpose
GraphControllerTemplate	Parses a class that inherits from *GraphController* to extract declared type extensions, query methods and mutation methods. *Note*: The controller template cannot directly create graph types, instead it acts as a container for adding fields to various virtual types (those not connected to developer code) created when a controller is added to a schema.
ObjectGraphTypeTemplate **InputObjectGraphTypeTemplate**	A general purpose template for parsing a model class to extract its methods and properties and generate fields for the type appropriately. ***Creates***: OBJECT, INPUT_OBJECT types
EnumGraphTypeTemplate	Parses a C# enumeration to generate appropriate values, deprecatable flags, descriptions etc. for the introspection system. ***Creates***: ENUM types
GraphDirectiveTemplate	Parses a class that inherits from GraphDirective to setup both the directive metadata as well as the resolvers to interact with the graph field pipeline as required. **Creates**: DIRECTIVE type
InterfaceGraphTypeTemplate	Parses a C# interface to generate a set of field stubs to put on the object graph. Interfaces are linked to actual graph types via the interfaces implemented on the C# classes that exist on the graph as an OBJECT type. ***Creates***: INTERFACE type

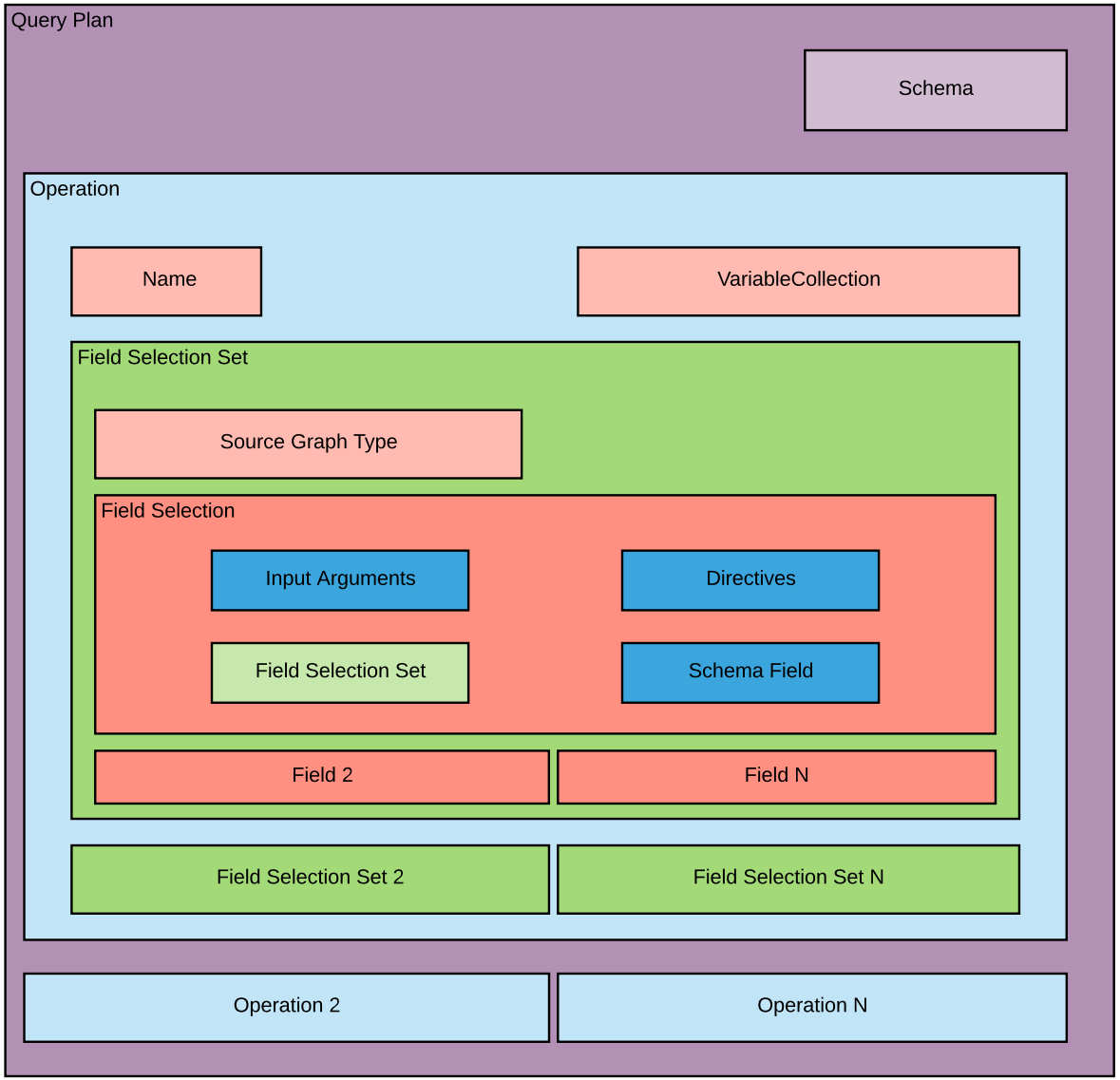


This document represents the top level templates used for generating an object graph. Not shown are the various templates used by these templates to convert methods and properties into qualified graph fields.

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

Query Plan Object Hierarchy

This document outlines a rough approximation of the structure of a Query Plan that is built from a user's graph query.



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.

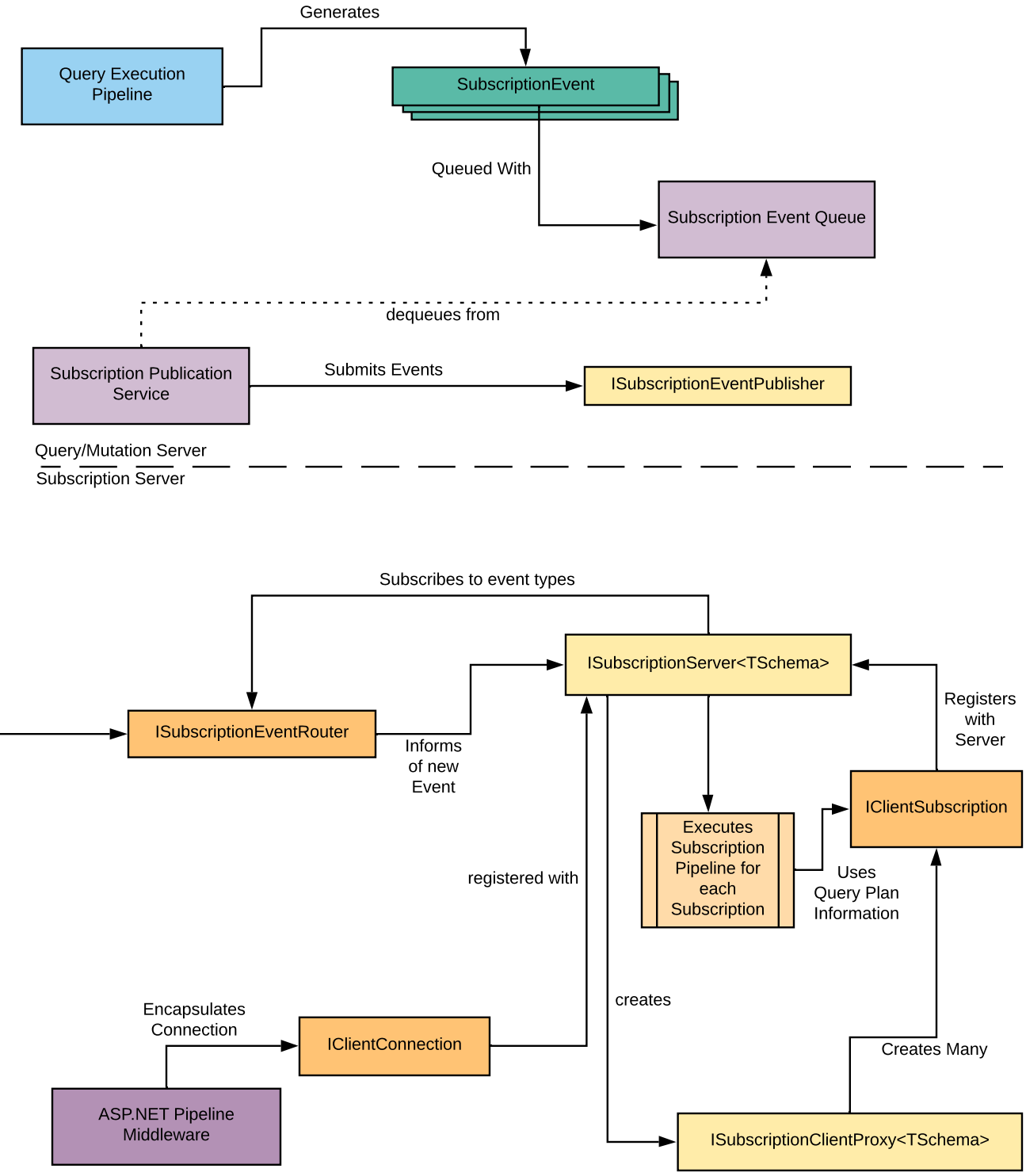
ISubscriptionClientProxy - Encapsulates the connection with GraphQL specifics (such as target schema) as well as the ability to monitor recieved events from the connection. The client proxy is "server specific" and can/should interprete messages to/from the client connection. For instance, `ApolloClientProxy` attempts to interprete any message from a client connection as a message that conforms to Apollo's graphql-over-websocket protocol.

IClientSubscription - Whenever the client proxy starts a new subscription (in whatever manner is appropriate for it), it should generate a new client subscription that can be inspected and handled by the server. The server uses these objects to deteremine if a recieved event should be routed to the client. This object may or may not be "server specific".

ISubscriptionServer - Primary object for handling received events. When an event is recieved the server is responsible for invoking the subscription pipeline to process the data, per registered subscription.

ISubscriptionEventRouter - The event router acts as an intermediary between the subscription server(s) and the event source. Some technology specific implementation is responsible for materializing SubscriptionEvents and delivering to the router for processing.

SubscriptionCollection - A helper object that provides useful tracking and filtering functions to a set of IClientSubscription objects. (Server Agnostic)



Apollo Server Components

This diagram shows the major objects that make up the out-of-the-box, in process subscription server. GraphQL ASP.NET implements Apollo's graphql-over-websocket protocol.