IXP Manager Workshop

28th Euro-IX Forum
April 24th 2016
Luxembourg

Grapher - Anatomy of a Request

Barry O'Donovan - INEX

barry.odonovan@inex.ie

Prologue

This slide deck was originally presented by Barry O'Donovan at the 28th Euro-IX Forum IXP Manager Workshop in Luxembourg.

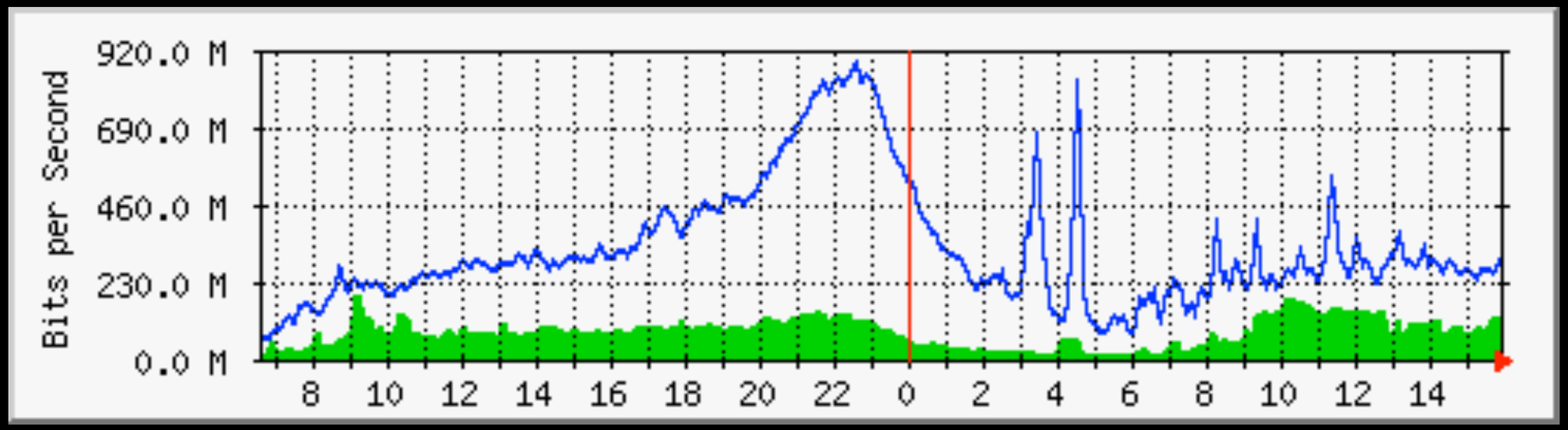
This presentation was a walk-through of how Grapher processes a request. There are essentially two sides to this: the HTTP request processing (including validation and access control) and the Grapher backend for processing a graph. I'm not sure how well this will translate as a standalone slide deck.

Grapher - Anatomy of a Request

- How does a request to:
  http://localhost:8088/grapher/phsyicalinterface?id=127
  turn into a graph?
- We're using the following configuration options:

```
GRAPHER_BACKENDS="sflow|mrtg|dummy"
GRAPHER_BACKEND_MRTG_LOGDIR="/vagrant/mrtg/v4"
GRAPHER_BACKEND_MRTG_WORKDIR="/vagrant/mrtg/v4"
GRAPHER_BACKEND_SFLOW_ROOT="/vagrant/sflow"
GRAPHER_CACHE_ENABLED=true
```

http://localhost:8088/grapher/phsyicalinterface?id=127



Service Providers

- Central place of all Laravel bootstrapping
- All providers registered in config/app.php

```
[ …, IXP\Providers\GrapherServiceProvider::class, … ]
```

- Builtins include: Auth, Cache, Database, Mail, Queue, Session, View
- Service providers register:
  - bindings (typically singletons)
  - event listeners
  - middleware
  - routes

IXP\Providers\GrapherServiceProvider

- Binds main service class:

```
$this->app->singleton( 'IXP\Services\Grapher', function($app) {
    return new \IXP\Services\Grapher;
});
```

- Registers *artisan* console commands
- Registers Grapher related view functions
  - e.g. scale() which formats floats to *23.3Mbps*
- Registers HTTP routes

IXP\Services\Grapher

- Single point of entry to all functionality
  - Resolves and instantiates backend(s)
  - Instantiates graph objects
  - Manages the graph cache
- Accessible via the service container or facade:

```
// $pi is an instance of a PhysicalInterface object

$grapher = $this->app->make('IXP\Services\Grapher');
$graph = $grapher->physint($pi);


$graph = Grapher::physint($pi);
```

Grapher - Anatomy of a Request

http://localhost:8088/grapher/phsyicalinterface

- Web server redirects to /path/to/ixp4/public/index.php
- index.php bootstraps and hands off to Laravel
- Laravel's HTTP kernel will look for a matching route for grapher/physicalinterface
- Routes set by Grapher service provider

Grapher - Anatomy of a Request

http://localhost:8088/grapher/phsyicalinterface

```
Route::group([ 'namespace'  => 'IXP\Http\Controllers\Services',
               'as'         => 'grapher::', 'prefix' => 'grapher',
               'middleware' => 'grapher'
          ],
          function(){
               // ...
               Route::get( 'phsyicalinterface', 'Grapher@physicalInterface' );
               // ...
          }
);
```

Grapher - Anatomy of a Request

http://localhost:8088/grapher/phsyicalinterface

```
Route::group([ 'namespace'  => 'IXP\Http\Controllers\Services',
               'as'         => 'grapher::', 'prefix' => 'grapher',
               'middleware' => 'grapher'
          ],
          function(){
               // ...
               Route::get( 'phsyicalinterface', 'Grapher@physicalInterface' );
               // ...
          }
);
```

- Namespace translates Grapher@physicalInterface to:
  - Class: IXP\Http\Controllers\Services\Grapher
  - Function: physicalInterface()

Grapher - Anatomy of a Request

http://localhost:8088/grapher/phsyicalinterface

```
Route::group([ 'namespace'  => 'IXP\Http\Controllers\Services',
               'as'         => 'grapher::', 'prefix' => 'grapher',
               'middleware' => 'grapher'
          ],
          function(){
               // ...
               Route::get( 'phsyicalinterface', 'Grapher@physicalInterface' );
               // ...
          }
);
```

- as - route name prefix for generating URLs

```
$url = route('grapher::physint');
```

Grapher - Anatomy of a Request

http://localhost:8088/grapher/phsyicalinterface

```
Route::group([ 'namespace'  => 'IXP\Http\Controllers\Services',
               'as'         => 'grapher::', 'prefix' => 'grapher',
               'middleware' => 'grapher'
          ],
          function(){
               // ...
               Route::get( 'phsyicalinterface', 'Grapher@physicalInterface' );
               // ...
          }
);
```

- prefix - URL prefix: localhost:8088/**grapher**

Grapher - Anatomy of a Request

http://localhost:8088/grapher/phsyicalinterface

```
Route::group([ 'namespace'  => 'IXP\Http\Controllers\Services',
               'as'         => 'grapher::', 'prefix' => 'grapher',
               'middleware' => 'grapher'
          ],
          function(){
               // ...
               Route::get( 'phsyicalinterface', 'Grapher@physicalInterface' );
               // ...
          }
);
```

- HTTP middleware is a mechanism for filtering requests
  - *grapher* middleware defined in IXP\Http\Kernel::$routeMiddleware
    `\IXP\Http\Middleware\Services\Grapher::class`

Grapher Middleware

- Preprocesses all HTTP *grapher/** requests

```php
public function handle($request, Closure $next ) {
// get the grapher service
$grapher = App::make('IXP\Services\Grapher');

// all graph requests require a certain basic set of
// parameters / defaults. Let's take care of that here
$graph = $this->processParameters( $request, $grapher );
```

Grapher Middleware

```
private function processParameters( Request $request,
    GrapherService $grapher ): Graph {

  // …
```

```
$request->period = Graph::processParameterPeriod(
    $request->input( 'period',   '' )
);

$request->category = Graph::processParameterCategory(…);
$request->protocol = Graph::processParameterProtocol(…);
$request->type     = Graph::processParameterType(…);


// Graph => IXP\Services\Grapher\Graph
```

Grapher Middleware

```
private function processParameters( Request $request,
    GrapherService $grapher ): Graph {

  // …
```

Let's take a quick look at this

```
$request->period = Graph::processParameterPeriod(
    $request->input( 'period',   '' )
);
```

```
$request->category = Graph::processParameterCategory(…);
$request->protocol = Graph::processParameterProtocol(…);
$request->type     = Graph::processParameterType(…);


// Graph => IXP\Services\Grapher\Graph
```

Graph::processParameterPeriod()

```
const PERIOD_DAY   = 'day';
// PERIOD_WEEK, MONTH, YEAR
const PERIOD_DEFAULT  = self::PERIOD_DAY;

const PERIODS = [
  self::PERIOD_DAY   => self::PERIOD_DAY,
  self::PERIOD_WEEK  => self::PERIOD_WEEK,
  self::PERIOD_MONTH => self::PERIOD_MONTH,
  self::PERIOD_YEAR  => self::PERIOD_YEAR
];

private $period = self::PERIOD_DEFAULT;

public static function processParameterPeriod( string $v ): string {
  if( !isset( self::PERIODS[ $v ] ) ) {
    $v = self::PERIOD_DEFAULT;
  }
  return $v;
}
```

Grapher Middleware

```
private function processParameters( Request $request,
    GrapherService $grapher ): Graph {

    // …
```

Period validated / sanitised and re-injected into $request

```
$request->period = Graph::processParameterPeriod(
    $request->input( 'period',   '' )
);
```

```
$request->category = Graph::processParameterCategory(…);
$request->protocol = Graph::processParameterProtocol(…);
$request->type     = Graph::processParameterType(…);


// Graph => IXP\Services\Grapher\Graph
```

Grapher Middleware

```
use IXP\Services\Grapher\Graph\PhysicalInterface as PhysIntGraph;

private function processParameters( Request $request,
    GrapherService $grapher ): Graph {

  // params processed. let's continue: $target is taken from the url
  switch( $target ) {
    // …
    case 'phsyicalinterface':
      $physint = PhysIntGraph::processParameterPhysicalInterface(
          (int)$request->input( 'id', 0 ) );
      $graph = $grapher->physint( $physint )
          ->setParamsFromArray( $request->all() );
      break;
  // …
  }

  return $graph;
}
```

Grapher Middleware

```
use IXP\Services\Grapher\Graph\hysicalInterface as PhysIntGraph;

private function processParameters( Request $request,
    GrapherService $grapher ): Graph {

  // parameters processed…

  switch( $target ) {
    // …
    case 'phsyicalinterface':
      $physint = PhysIntGraph::processParameterPhysicalInterface(
          (int)$request->input( 'id', 0 ) );
      $graph = $grapher->physint( $physint )
          ->setParamsFromArray( $request->all() );
      break;
  // …
  }

  return $graph;
}
```

Let's take a quick look at this

PhysicalInterfaceGraph ::processParameterPhysicalInterface()

```
public static function processParameterPhysicalInterface( int $pi )
      : PhysicalInterfaceEntity {


  if( !$pi
     || !( $physint = d2r( 'PhysicalInterface' )->find( $pi ) ) ) {
            abort(404);
    }

  return $physint;
}
```

Loads the physical interface from the DB is the id ($pi) exists. Otherwise throws a 404.

Grapher Middleware

```
use IXP\Services\Grapher\Graph\hysicalInterface as PhysIntGraph;

private function processParameters( Request $request,
    GrapherService $grapher ): Graph {

  // parameters processed…

  switch( $target ) {
    // …
    case 'phsyicalinterface':
      $physint = PhysIntGraph::processParameterPhysicalInterface(
          (int)$request->input( 'id', 0 ) );
      $graph = $grapher->physint( $physint )
          ->setParamsFromArray( $request->all() );
      break;
  // …
  }

  return $graph;
}
```

Now we have a valid Graph object!

Grapher Middleware

```
use IXP\Services\Grapher\Graph\hysicalInterface as PhysIntGraph;

private function processParameters( Request $request,
    GrapherService $grapher ): Graph {

  // parameters processed…

  switch( $target ) {
    // …
    case 'phsyicalinterface':
      $physint = PhysIntGraph::processParameterPhysicalInterface(
          (int)$request->input( 'id', 0 ) );
      $graph = $grapher->physint( $physint )
          ->setParamsFromArray( $request->all() );
      break;
  // …
  }

  return $graph;
}
```

Grapher Middleware

- Preprocesses all HTTP *grapher/** requests

```
public function handle($request, Closure $next ) {
    // get the grapher service
    $grapher = App::make('IXP\Services\Grapher');

    // all graph requests require a certain basic set of
    // parameters / defaults. Let's take care of that here
    $graph = $this->processParameters( $request, $grapher );
```

$graph returned back to here.

Grapher Middleware

- Preprocesses all HTTP *grapher/** requests

```
public function handle($request, Closure $next ) {
    // …
    $graph = $this->processParameters( $request, $grapher );

    // so we know what graph we need and who's looking for it
    // let's authorise for access (this throws an exception)
    $graph->authorise();

    $request->attributes->add(['graph' => $graph]);

    return $next($request);
}
```

Let’s take a quick look at this

$graph->authorise()

```
public function authorise(): bool {
        if( !Auth::check() ) { return $this->deny(); }

        if( Auth::user()->isSuperUser() ) { return $this->allow(); }

        if( Auth::user()->getCustomer()->getId() ==
            $this->physicalInterface()
                ->getVirtualInterface()
                ->getCustomer()
                ->getId() ) {
            return $this->allow();
        }

        Log::notice( … );
        return $this->deny();
    }
```

$graph->authorise()

```
public function authorise(): bool {
        if( !Auth::check() ) { return $this->deny(); }

        if( Auth::user()->isSuperUser() ) { return $this->allow(); }

        if( Auth::user()->getCustomer()->getId() ==
            $this->physicalInterface()
                ->getVirtualInterface()
                ->getCustomer()
                ->getId() ) {
            return $this->allow();
        }

        Log::notice( … );
        return $this->deny();
}
```

Default: throws Illuminate\Auth\Access\AuthorizationException

Grapher Middleware

- Preprocesses all HTTP *grapher/** requests

```
public function handle($request, Closure $next ) {
    // …
    $graph = $this->processParameters( $request, $grapher );

    // so we know what graph we need and who's looking for it
    // let's authorise for access (this throws an exception)
    $graph->authorise();

    $request->attributes->add(['graph' => $graph]);

    return $next($request);
}
```

Grapher Controller

- Control is passed to the appropriate controller once *all* middleware has completed.

```
Route::get( 'phsyicalinterface', 'Grapher@physicalInterface' );
```

- All graphs handled generically right now:

```
class Grapher extends Controller
{
  public function __construct( Request $request, … ) {
    $this->graph  = $request->attributes->get('graph');
  }

  public function physicalInterface( Request $request ): Response {
    return $this->simpleResponse( $request );
  }
}
```

Grapher Controller

```
private function simpleResponse( $request ): Response {
   return (new Response( call_user_func([$this->graph(),$this->graph()->type()])))
    ->header('Content-Type', Graph::CONTENT_TYPES[ $this->graph()->type() ] )
    ->header('Content-Disposition', sprintf( 'inline; filename="xxx"' ) )
    ->header( 'Expires', Carbon::now()->addMinutes(5)->toRfc1123String() );
}
```

Grapher Controller

```
private function simpleResponse( $request ): Response {
  return (new Response( call_user_func([$this->graph(),$this->graph()->type()])))
   ->header('Content-Type', Graph::CONTENT_TYPES[ $this->graph()->type() ] )
   ->header('Content-Disposition', sprintf( 'inline; filename="xxx"' ) )
   ->header( 'Expires', Carbon::now()->addMinutes(5)->toRfc1123String() );
}
```

```
new Response(
   call_user_func(
      [ $this->graph(), $this->graph()->type() ]
   )
)
```

Grapher Controller

```
private function simpleResponse( $request ): Response {
  return (new Response( call_user_func([$this->graph(),$this->type()])))
   ->header('Content-Type', Graph::CONTENT_TYPES[ $this->graph()->type() ] )
   ->header('Content-Disposition', sprintf( 'inline; filename="xxx"' ) )
   ->header( 'Expires', Carbon::now()->addMinutes(5)->toRfc1123String() );
}
```

```
new Response(
   call_user_func(
      [ $this->graph(), $this->graph()->type() ]
   )
)
```

e.g. `$this->graph()->png()`

Graph Objects

- IXP\Services\Grapher\Graph is an abstract class which:
  - defines all common parameters and related functions (period, etc)
    - accessors, setters, parameters processors
  - is provided a backend by Grapher
  - has other objects such as statistics(), renderer(), data()
  - utility functions such as toc(), url()
  - key fns: data(), png(), rrd(), log() [json'ified data()], json() [of toc()]
  - presentation fns: identifier(), name(), title(), watermark()
  - access control: authorise(), allow(), deny()
- All concrete graph implementations extend this class

Graph Objects :: Physical Interface

- Extends IXP\Services\Grapher\Graph and:
  - defines accessors/getters/parameter processors for
    Entities\PhysicalInterface
  - overrides name(), identifier(), url()
  - overrides authorise()

Graph Objects :: Create a PNG

```
$this->graph()->png()
```

Graph Objects :: Create a PNG

```
$this->graph()->png()
```

```
public function png(): string {
   return $this->grapher()->remember(
      $this->cacheKey('png'), function() {
         return $this->backend()->png($this);
      }
   );
}
```

Graph Objects :: Create a PNG

```
$this->backend()->png($this)
```

```
public function backend(): GrapherBackend {
   if( $this->backend === null ) {
      $this->backend = $this->grapher()->backendForGraph( $this );
   }
   return $this->backend;
}
```

Graph Objects :: Create a PNG

```
$this->grapher()->backendForGraph( $this )
```

```
public function backendForGraph( Graph $graph, array $backends = [] )
        : BackendContract {
  if( !count( $backends ) ) {
    $backends = config('grapher.backend');
  }

  if( !count( $backends ) ) {
    throw new ConfigurationException(…);
  }

  foreach( $backends as $backend ) {
    if( ( $b = $this->backend( $backend ) )->canProcess( $graph ) ) {
      return $b;
    }
  }

  throw new GraphCannotBeProcessedException(…);
}
```

Graph Objects :: Create a PNG

inex internet neutral exchange

$this**->grapher()->backendForGraph(**$this)

```
public function backendForGraph( Graph $graph, array $backends = [] )
        : BackendContract {
  if( !count( $backends ) ) {
    $backends = config('grapher.backend');
  }

  if( !count( $backends ) ) {
    throw new ConfigurationException(…);
  }

  foreach( $backends as $backend ) {
    if( ( $b = $this->backend( $backend ) )->canProcess( $graph ) ) {
      return $b;
    }
  }

  throw new GraphCannotBeProcessedException(…);
}
```

Graph Objects :: Create a PNG

inex internet neutral exchange

```
$this->backend( $backend ) )->canProcess( $graph )
```

```
public function canProcess( Graph $graph ): bool {
  // find what this backend can support
  $s = $this->supports();

  if( isset( $s[ $graph->lcClassType() ] )
    && ( isset($s[ $graph->lcClassType() ]['categories'])
      && in_array( $graph->category(),
            $s[ $graph->lcClassType() ]['categories'] ) )
    && // periods
    && // protocols
    && // types ) )
  ) {
    return true;
  }

  return false;
}
```

Graph Objects :: Create a PNG

```
$this->backend()->png($this)
```

```
public function backend(): GrapherBackend {
   if( $this->backend === null ) {
      $this->backend = $this->grapher()->backendForGraph( $this );
   }
   return $this->backend;
}
```

Graph Objects :: Create a PNG

inex internet neutral exchange

```
$this->graph()->png()
```

```
public function png(): string {
    return $this->grapher()->remember(
        $this->cacheKey('png'), function() {
            return $this->backend()->png($this);
        }
    );
}
```

Backend Objects

- IXP\Services\Grapher\Backend is an abstract class which:
  - defines the canProcess() implementation
- All concrete backend implementations extend this class **and implement IXP\Contracts\Grapher\Backend** which requires certain functions to be implemented:
  - name()
  - isConfigurationRequired() / generateConfiguration()
  - canProcess()
  - data(), png()
  - supports()

Backend Objects :: supports()

inex internet neutral exchange

```
public static function supports(): array {
   return [
      // ...
      'physicalinterface' => [
        'protocols'    => [ Graph::PROTOCOL_ALL => Graph::PROTOCOL_ALL ],
        'categories'   => Graph::CATEGORIES,
        'periods'      => Graph::PERIODS,
        'types'        => array_except( Graph::TYPES, Graph::TYPE_RRD )
      ],
      // ...
   ];
}
```

Backend Objects :: png()

```
public function png( Graph $graph ): string {
  if( ( $img = @file_get_contents(
          $this->resolveFilePath( $graph, 'png' ) ) ) === false ){
    // couldn't load the image so return a placeholder
    Log::notice(..);
    return @file_get_contents( "…/image-missing.png" );
  }

  return $img;
}
```

Backend Objects :: png()

```
public function png( Graph $graph ): string {
  if( ( $img = @file_get_contents(
          $this->resolveFilePath( $graph, 'png' ) ) ) === false ){
    // couldn't load the image so return a placeholder
    Log::notice(..);
    return @file_get_contents( "…/image-missing.png" );
  }

  return $img;
}
```

IXP Manager Workshop

28th Euro-IX Forum
April 24th 2016
Luxembourg

Grapher - Anatomy of a Request

Barry O'Donovan - INEX

barry.odonovan@inex.ie