Apache Wink Developer Guide

Software Version: 1.0

Draft Version

(This document is still under construction)

Document Release Date: [August 2009]

Software Release Date: [August 2009]

This page last changed on Aug 09, 2009 by michael.

Apache Wink

Apache Wink is a complete Java based solution for implementing and consuming REST based Web Services. The goal of the Apache Wink framework is to provide a reusable and extendable set of classes and interfaces that will serve as a foundation on which a developer can efficiently construct applications.

Contents

1 Introduction to Apache Wink

This page last changed on Aug 04, 2009 by michael.

Introduction to Apache Wink

Apache Wink 1.0 is a complete Java based solution for implementing and consuming Rest based Web Services. The goal of the Wink framework is to provide a reusable and extendable set of classes and interfaces that will serve as a foundation on which a developer can efficiently construct applications.

Wink consists of a Server module for developing Rest services, and of a Client module for consuming Rest services. It cleanly separates the low-level protocol aspects from the application aspects. Therefore, in order to implement and consume Rest Web Services the developer only needs to focus on the application business logic and not on the low-level technical details.
The Wink Developer Guide provides the developer with a rudimentary understanding of the Wink framework and the building blocks that comprise it.

Welcome to Apache Wink

Wink is a framework for the simple implementation and consumption of Rest web services. Rest is an acronym that stands for REpresentational State Transfer. Rest web services are "Resources" that are identified by unique URIs. These resources are accessed and manipulated using a set of "Uniform methods". Each resource has one or more "Representations" that are transferred between the client and the service during a web service invocation.

The central features that distinguish the Rest architectural style from other network-based styles is its emphasis on a uniform interface, multi representations and services introspection.
Wink facilitates the development and consumption of Rest web services by providing the means for modeling the service according to the Rest architectural style. Wink provides the necessary infrastructure for defining and implementing the resources, representations and uniform methods that comprise a service.

Rest Architecture

For a detailed understanding of the Rest architecture refer to the description by Roy Fielding in his dissertation, The Design of Network-based Software Architectures
www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Rest Web Service

Figure 1: Rest Web service design structure

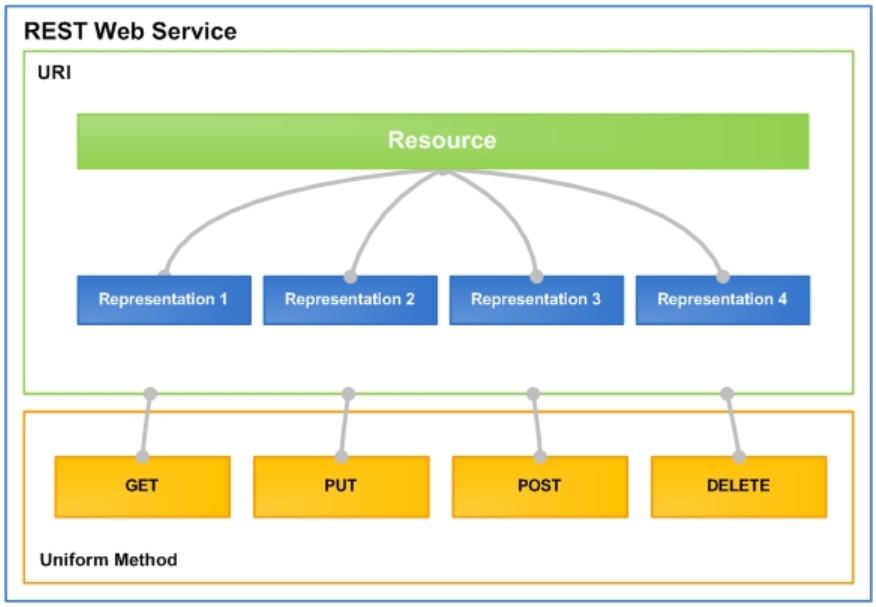


Figure 1 demonstrates the design principles and components that comprise a Rest web service. Wink reflects these design principles in the implementation of web services.

Apache Wink Open Development

The purpose of this document is to provide detailed information about Wink 1.0 and describe the additional features that the Wink 1.0 runtime provides in addition to the JAX-RS Java API for REST Web Service specification.

In addition to the features description, this document also provides information regarding implementation specific issues.
This document provides the developer with a rudimentary understanding of the Wink 1.0 framework in order to highlight the underlying concepts and precepts that make up the framework in order to create a basis for understanding, cooperation and open development of Wink.

JAX-RS Specification Document
For more information on the JAX-RS functionality, refer to the JAX-RS specification document, available at the following location:
http://jcp.org/aboutJava/communityprocess/final/jsr311/index.html

JAX-RS Compliancy

Wink 1.0 is a complete implementation of the JAX-RS v1.0 specification.

The JAX-RS TCK tests still need to performed in order to be able to declare that it is JAX-RS compliant. JAX-RS is a Java based API for RESTful Web Services is a Java programming language API that provides support in creating web services according to the Representational State Transfer (REST) architectural style. JAX-RS uses annotations, introduced in Java SE 5, to simplify the development and deployment of web service clients and endpoints.

2 Apache Wink Building Blocks

This page last changed on Aug 04, 2009 by michael.

In order to take full advantage of Apache Wink 1.0, a basic understanding of the building blocks that comprise it and their functional integration is required. The following chapter describes the basic concepts and building blocks of Wink, version 0.1.

This chapter contains the following sections

Service Implementation Building Block Overview

As mentioned in the previous chapter, Apache Wink 1.0 reflects the design principles of a REST web service. It does so by providing the developer with a set of java classes that enable the implementation of "**Resources**", "**Representations**" and the association between them. Wink 1.0 also enables the developer to define the resource URI and the "**Uniform methods**" that are applicable to the resource.

Resource

A "**resource**" represents a serviceable component that enables for the retrieval and manipulation of data. A "**resource class**" is used to implement a resource by defining the "**resource methods**" that handle requests by implementing the business logic. A resource is bound or anchored to a URI space by annotating the resource class with the @Path annotation.

Providers

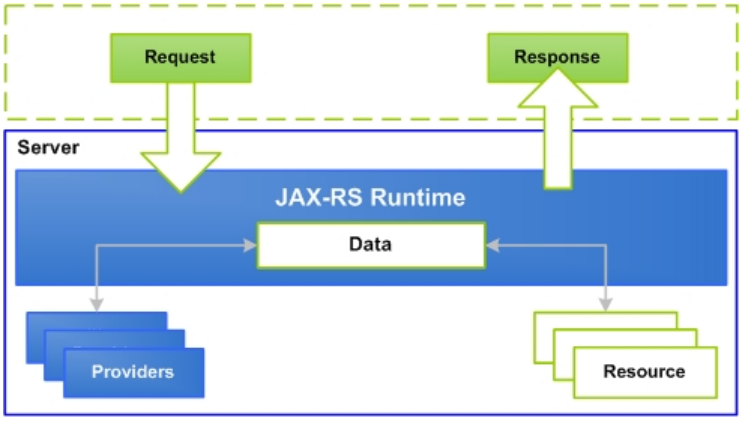
A provider is a class that is annotated with the @Provider annotation and implements one or more interfaces defined by the JAX-RS specification. Providers are not bound to any specific resource. The appropriate provider is automatically selected by the Apache Wink runtime according to the JAX-RS specification.
There are three types of providers defined by the JAX-RS specification:

- Entry Providers
- Context Providers
- Exception Mapping Provider

Entity Provider

An "**Entity Provider**" is a class that converts server data into a specific format requested by the client and or converts a request transmitted by the client into server data. An entity provider can be restricted to support a limited set of media types using the @Produces and @Consumes annotations. An entity provider is configured to handle a specific server data type by implementing the MessageBodyWriter and or MessageBodyReader interfaces.

Figure 2: Entity Provider Diagram



Context Provider

Context providers are used to supply contexts to resource classes and other providers by implementing the context ContextResolver interface. Context providers may restrict the media types that they support using the @Produces annotation.

Figure 3: Context Provider Diagram
TBD (Picture)

Exception Mapping Provider

Exception mapping providers map a checked or runtime exception into an instance of a Response by implementing the ExceptionMapper interface. When a resource method throws an exception for which there is an exception mapping provider, the matching provider is used to obtain a Response instance.

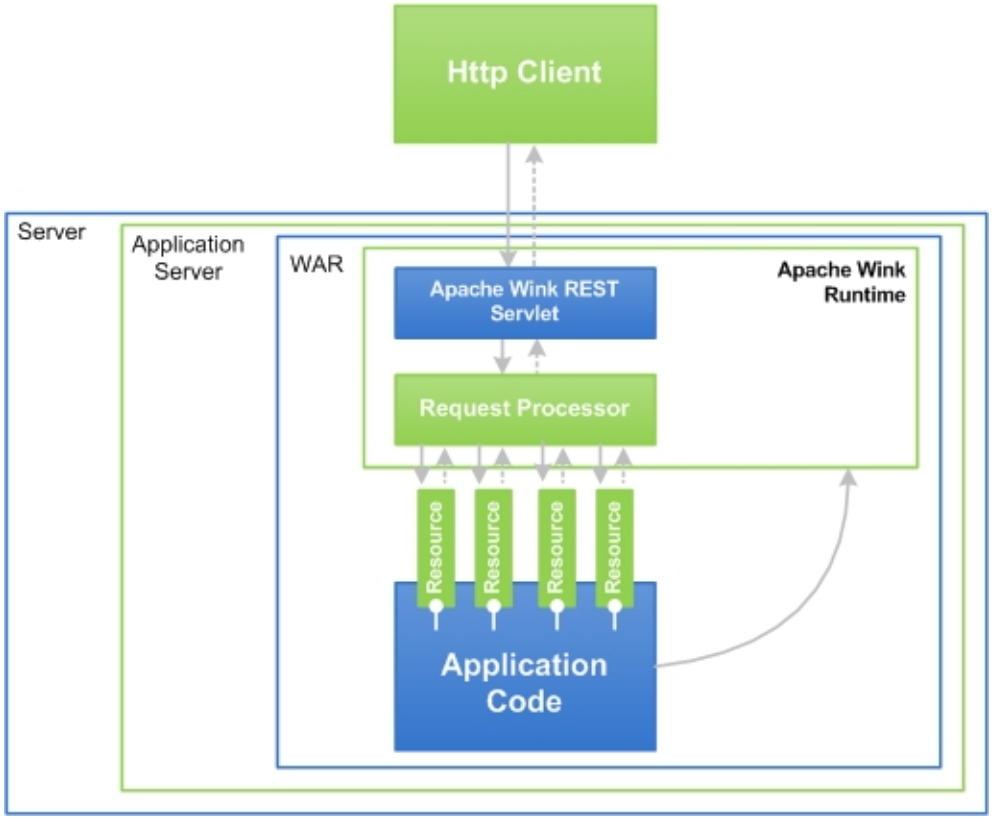
Figure 4: Exception Mapping Provider Diagram
TBD (Picture)

URI Dispatching

Designing an efficient REST web service requires that the application developer understands the resources that comprise the service, how to best identify the resources, and how they relate to one another.

Restful resources are identified by URIs in most cases related resources have URIs that share common path elements.

Figure 5: Apache Wink Logic Flow



Symphony SDK Logic Flow

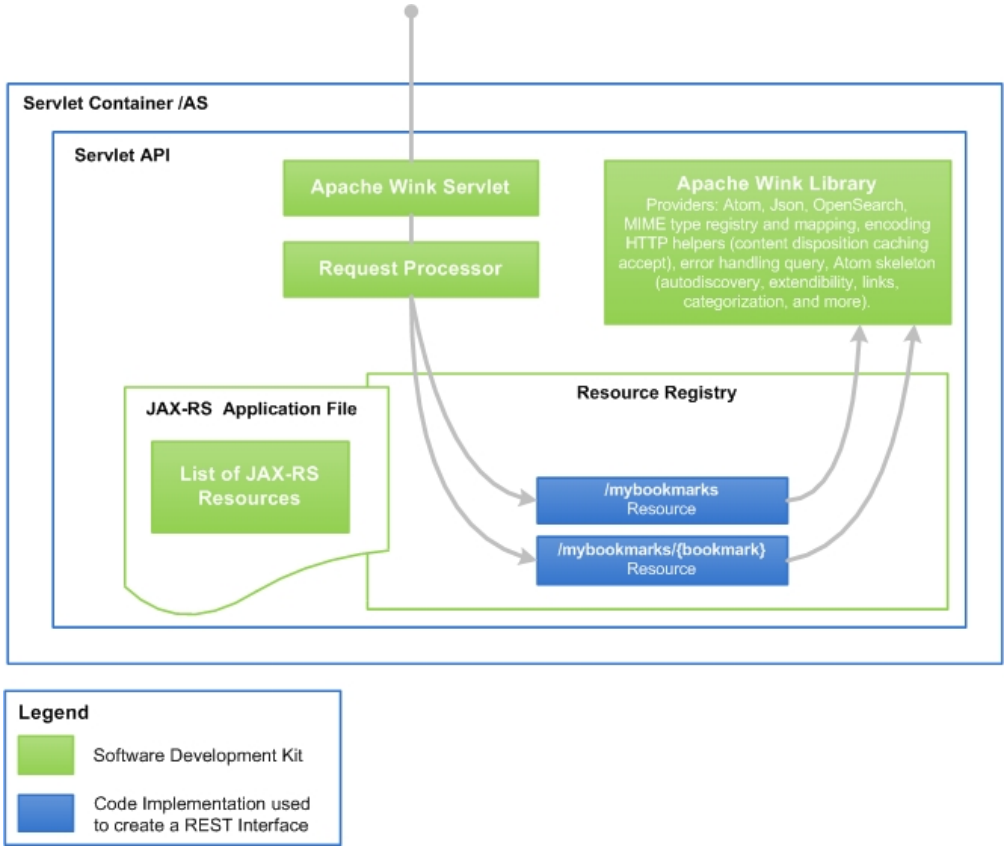
Figure 5 illustrates the Apache Wink logic flow. The Http request sent by the client invokes the "**Apache Wink Rest Servlet**". The Rest servlet uses the "**Request Dispatcher**" and the request URI in order to find, match and invoke the correct resource method.

Bookmarks Example

Throughout this document, various project examples are used in order to describe the functionality and processes that comprise the Apache Wink.
In order to explain the Rest design principles used in the Apache Wink this developer guide refers to the "**Bookmark**" example project found in the examples folder located in the Apache Wink distribution.
Refer to the code (using an IDE application) in the example in conjunction with the following explanations and illustrations in this developer guide.

Apache Wink Servlet and Request Processor

Figure 6: Apache Wink Rest Servlet and Request Processor for the Bookmark Services



Server and Reqest Processor

Figure 6 shows the Apache Wink servlet and request Processor concept in the context of the application server. In the "**Bookmarks**" example in figure 6 there are two Resources, the first Resource is assosiated with the **/mybookmarks** URI and manages the bookmarks collection, the second resource is assosiated with the **/mybookmarks/{bookmark}** Resources and manages an individual bookmark within the collection.

The Resources' defined by the web service and managed by Apache Wink are referred to as **"URI space"**. The Resources space is the collection of all URI's that exist in the same context. Figure 6 shows the URI spacew that contains **/mybookmarks** and **/mybookmarks/{bookmarks}**.

URI Space

The Bookmarks service URI space consists of the following URI space items and detailed descriptions about their context and functionality.
Table 1: URI Management

URI space Item	Description
/Bookmark/rest	This URI is the root context of the bookmark service and the entry point of the URI space of the service. An Http GET request to this URI returns a "Service Document" which is automatically generated by Apache Wink. The service document provides information about all available collections in the URI space.
/Bookmark/rest /mybookmarks	This URI is associated with a collection of bookmarks resources. Clients use the Http GET method in order to retrieve a representation of

	the collection and Http POST method in order to create a new item in the collection.
/Bookmark/rest /mybookmarks/ {bookmark}	This URI template is associated with a single bookmark resource. Clients use the Http GET method in order to retrieve a representation of the resource, Http PUT method is used in order to update the resource and Http DELETE method is used in order to delete the resource.

Assets

Assets are classes that contain "**web service business logic**" implemented by the developer. Each Asset is associated with one or more URI. The Apache Wink dispatcher invokes the Asset, which is associated with the URI found in the Http request.
An Asset class can implement one or more methods, each method is associated with a single Http method (GET, HEAD, POST, PUT, DELETE etc). The methods can be associated with a Mime type of a produced representation. Methods that handle Http verbs of requests with a body (such as PUT, POST) are also associated with the Mime type of the Http request.
The Asset class can be registered to the Apache Wink using the "Spring context xml" or by using a registration API.

> For further information regarding the Spring Context, refer to the "**Advanced Functionality**" chapter xx, and go to the "**Spring Context Configuration**" section, on page xx.

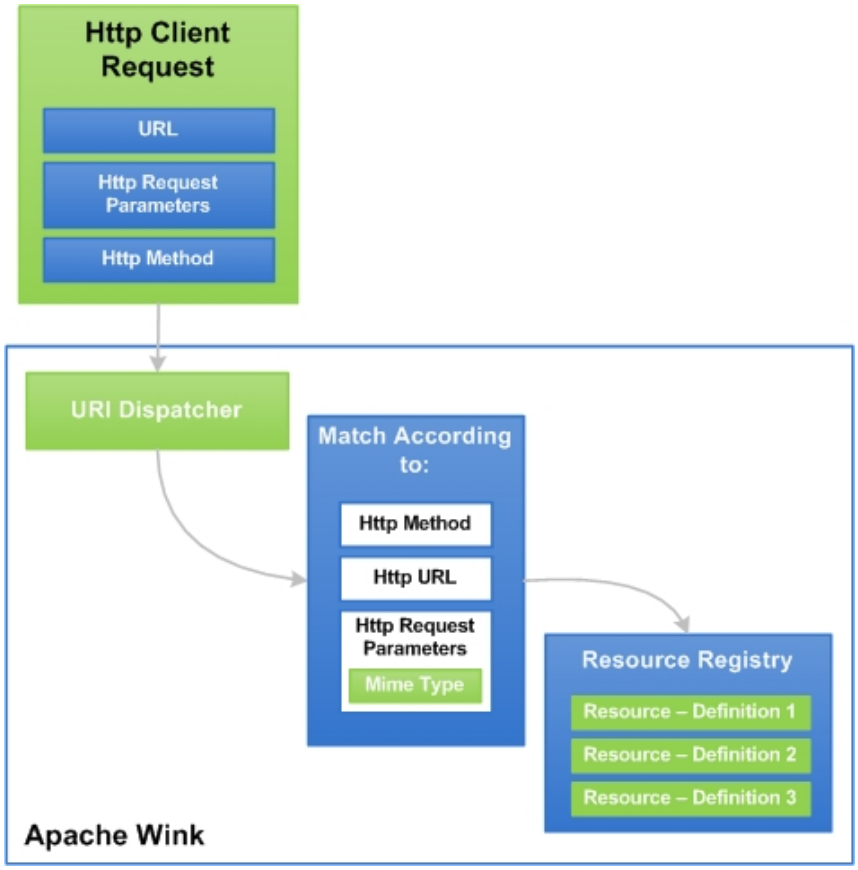
Annotations

TBD

URL Handling

The Apache Wink receives Http requests and then dispatches a wrapped Http request to the appropriate Resource method.
The Http request is match to the Resource method based on the Http request parameters, the Resource method definitions and the Mime type.

Figure 7: URL Request Handling



Request Handling

Figure 7 demonstrates the Http Client request path to the URI dispatcher, once the dispatcher receives the request it is then matched according to the Http method, URL and Mime type and finally the Resource registry definition.

Http Methods - GET, POST, PUT, DELETE and OPTIONS

The common Http 1.1 methods for the Apache Wink are defined in the following section. This set of methods can be expanded.

Method Usage

Table 3: Http Methods

Method	Safe	Idempotent	Cacheable
GET	X	X	X
HEAD	X	X	X
PUT		X	
POST			*
DELETE		X	
OPTIONS			

Key - X

- **Safe** - does not affect the server state
- **Idempotent** - a repeated application of the same method has the same effect as a single application
- **Cacheable** - a response to a method is cacheable if it meets the requirements for Http cachin
- ***** - Responses to this method are not cacheable, unless the response includes an appropriate Cache-Control or Expires header fields. However, the 303 response can be used to direct the user agent to retrieve a cacheable resource.

GET

The GET method is used to retrieve information from a specified URI and is assumed to be a safe and repeatable operation by browsers, caches and other Http aware components. This means that the operation must have no side effects and GET method requests can be re-issued.

HEAD

The HEAD method is the same as the GET method except for the fact that the HEAD does not contain message body.

POST

The POST method is used for operations that have side effects and cannot be safely repeated. For example, transferring money from one bank account to another has side effects and should not be repeated without explicit approval by the user.
The POST method submits data to be processed, for example, from an Html form, to the identified resource. The data is included in the body of the request. This may result in the creation of a new resource or the updates of existing resources or both.

PUT

The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity should be considered as a modified version of the one residing on the origin server.
If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI.

DELETE

The DELETE method requests that the origin server delete the resource identified by the Request-URI. This method can be overridden on the origin server.
The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully.

OPTIONS

The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI.
This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.

Basic URL Query Parameters

A URL parameter is a name and value pair appended to a URL. The parameter begins with a question mark "**?**" and takes the form of name=value.
If more than one URL parameter exists, each parameter is separated by an ampersand "**&**" symbol. URL parameters enable the client to send data as part of the URL to the server.
When a server receives a request and parameters are appended to the URL of the request, the server uses these parameters as if they were sent as part of the request body. There are several predefined URL

parameters recognized by the Symphony SDK. The following table lists the parameters commonly used in web service URLs. These special URL parameters are defined in the "**RestConstants**" class.

Query Parameters

Table 4: URL Parameters

Parameter	Description	Value
alt	Provides an alternative representation of the specified Mime type. Apache Wink recognizes this as a representation request of the highest priority.	Mime type, e.g. "text%2Fplain"
absolute-urls	Indicates to Apache Wink that the generated links in the response should be absolute, mutual exclusive with the relative-urls parameter	NONE
relative-urls	Indicates to Apache Wink that the generated links in the response should be relative, mutual exclusive with the absolute-urls parameter	NONE
callback	Wrap javascript representation in callback function, is relevant when requested with an application/json MimeType.	name of callback function. For example, "myfunc"

Combining URL Parameters

A single URL can contain more than one URL parameter, example **"?alt=text%2Fjavascript &callback=myfunc"**(where "%2F" represents escaped "/").

Apache Wink Building Blocks Summary

The previous section "**Service Implementation Building Blocks**" outlines the basic precepts and building blocks that comprise the service side of Apache Wink.
In order to understand the relationship between the building blocks that comprise Apache Wink a set of example applications have been designed and built that provide a reference point that demonstrate a rudimentary understanding about the functionality of Apache Wink.

Apache Wink Examples

The following examples applications are used in this "**Apache Wink Developer Guide**".

- Bookmarks
- HelloWorld
- QADefects

Bookmarks Project

This developer guide uses the bookmarks example application in order to describe the logic flow process within Apache Wink.
Refer to the comments located in the "**Bookmarks**" example application code for in-depth explanations about the methods used to build the bookmarks application.

HelloWorld Project

Complete the step-by-step "**HelloWorld**" tutorial in chapter 3 "**Getting Started with Apache Wink**" and then follow the installation instructions on page xx, in order to view the "**Bookmarks**" example application from within the Eclipse IDE.

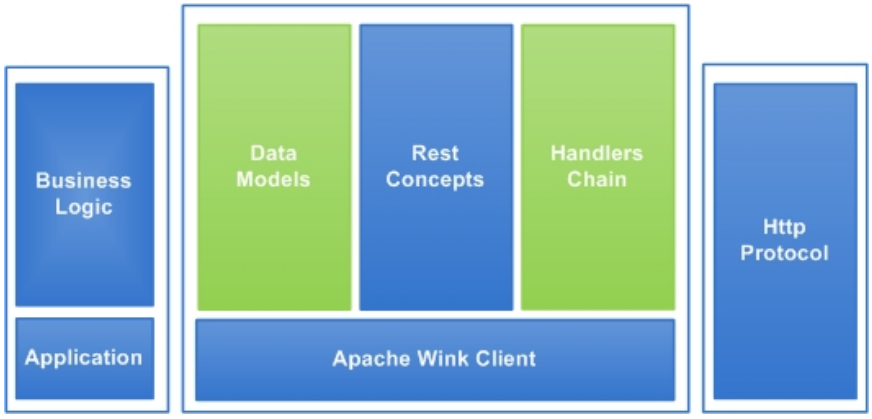
QADefects

The QADefects example application illustrates the advanced functionality of Apache Wink by implementing most of the features provided by the Apache Wink (Runtime) framework.

Client Component Basics Overview

Apache Wink interacts with Rest Web-Services. It maps Rest concepts to Java classes and encapsulates underlying Rest related standards and protocols, as well as providing a plug-in mechanism for RAW Http streaming data manipulation. This mechanism also provides the ability to embed cross application functionality on the client side, such as security, compression and caching.

Figure 8: Apache Wink Client Simplified Breakdown



Apache Wink Client

The illustration figure 8 shows the basic elements that comprise the client side of Apache Wink.

Apache Wink exposes several data models for use in the applications business logic. These data models are utilized by the application with the Rest concepts that the client exposes. Apache Wink provides the developer with the ability to implement and embed cross application functionality by implementing a Handler Chain mechanism. The Apache Wink Client communicates with Rest services via the Http protocol.

Client Components

Apache Wink is comprised of several key elements that together create a simple and convenient framework for the consumption of Rest based web services.

RestClient

The RestClient class is the central access point to Apache Wink. It provides the user with functionality that enables the creation of new resources. The RestClient provides the user with the ability to set different configuration parameters and propagated them, to each resource created. Once the resource is created, it can be manipulated by invoking the Http common methods, GET, PUT, POST and DELETE. The default usage of the Apache Wink is to create and reuse the RestClient object within the application.

Resource

TBD

ClientRequest

TBD

ClientResponse

TBD

ClientConfig

TBD

ClientHandler

TBD

InputStreamAdapter

TBD

EntityType

TBD

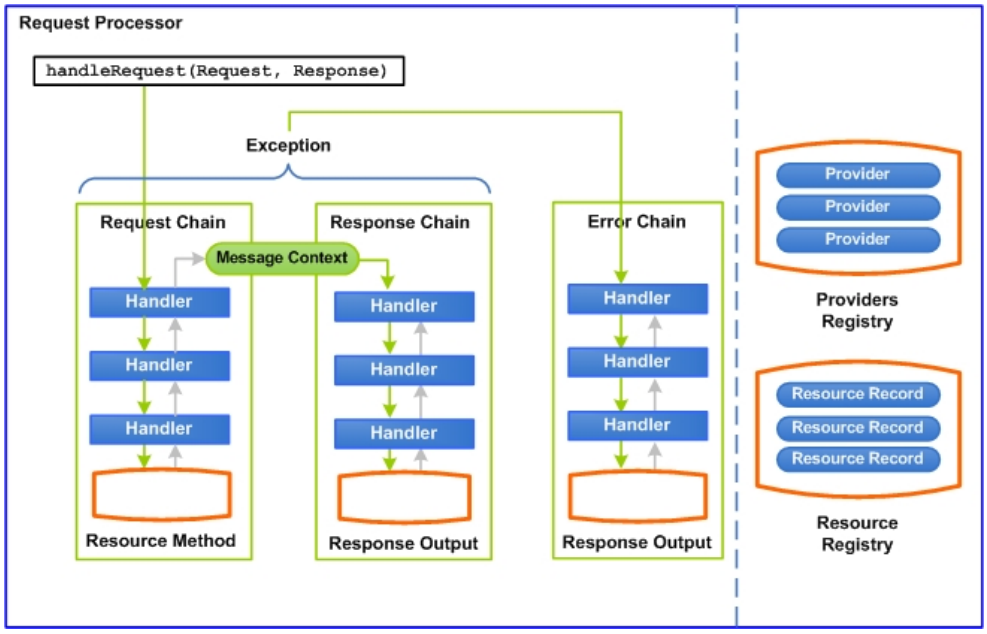
ApacheHttpClientConfig

TBD

The Apache Wink Runtime

The Apache Wink runtime is deployed on a JEE environment and is configured by defining the RestServlet in the web.xml file of the application. This servlet is the entry point of all the Http requests targeted for web services, and passes the request and response instances to the Symphony engine for processing.

Figure 9: Apache Wink Request Processor Architecture



The diagram illustrates the core components of the Apache Wink runtime. The Wink engine is the RequestProcessor. It builds an instance of a MessageContext with all of the required information for the request and passes it through the engine handler chains. The handler chains are responsible for serving the request, invoking the required resource method and finally generating a response.
In case of an error, the RequestProcessor invokes the Error chain with the generated exception for producing the appropriate response.
The Apache Wink runtime maintains providers and resources in two registries, the "providers registry" and the "resource registry" utilizing them during request processing.

Request Processor

The RequestProcessor is the Apache Wink engine, that is initialized by the RestServlet and is populated with an instance of a DeploymentConfiguration.
When a request is passed to the handleRequest() method of the RequestProcessor, a new instance of a MessageContext is created.
The MessageContext contains all of the information that is required for the Wink runtime to handle the request. The RequestProcessor first invokes the Request Handler Chain and then the Response Handler Chain.
If an exception occurs during any stage of the request processing, the RequestProcessor invokes the Error Handler Chain for processing the exception.

Deployment Configuration

The Apache Wink runtime is initialized with an instance of a Deployment Configuration. The Deployment Configuration holds the runtime configuration, including the handler chains, registries, configuration properties.
The Deployment Configuration is initialized with an instance of a JAX-RS Application used for obtaining user resources and providers.

Customization

The Deployment Configuration is customized by extending the DeplymentConfiguration class, overriding specific methods and specifying the new class in the web.xml file of the application.
In order to specify a different Deployment Configuration class instead of the default Deployment Configuration, the value of the symphony.deploymentConfiguration RestServlet init parameter must be set to be the fully qualified name of the customized configuration class.

```
<servlet>
  <servlet-name>restSdkService</servlet-name>
  <servlet-class>
    com.hp.symphony.server.internal.servlet.RestServlet
  </servlet-class>
  <init-param>
    <param-name>symphony.deploymentConfiguration</param-name>
    <param-value>com.hp.example.MyDeploymentConfig</param-value>
  </init-param>
</servlet>
```

The following table details the customizable methods of the Deployment Configuration class.

Deployment Configuration

Table 5: Deployment Configuration Customizable Methods

Method	Description
initAlternateShortcutMap	Initializes the AlternateShortcutMap. Refer to section TBD
initMediaTypeMapper	Initializes the MediaTypeMapper. Refer to section TBD
initRequestUserHandlers	Return a list of User Handler instances to embed in the Request chain. Refer to section TBD
initResponseUserHandlers	Return a list of User Handler instances to embed in the Response chain. Refer to section #TBD
initErrorUserHandlers	Return a list of User Handler instances to embed in the Error chain. Refer to section #TBD

Handler Chains

The handler chain pattern is used by the Wink runtime for implementing the core functionalities. There are three handler chains utilized by the Wink runtime:

- RequestHandlersChain
- ResponseHandlersChain
- ErrorHandlersChain

Refer to Chapter ????*TBD*???? Reference source not found. for more information on Handler Chains.

Registries

The Apache Wink runtime utilizes two registries for maintaining the JAX-RS resources and providers. Both registries maintain their elements in a sorted state according to the JAX-RS specification for increasing performance during request processing. In addition to the JAX-RS specification sorting, Wink supports the prioritization of resources and providers.

Refer to section ??*TBD*?? for more information on Resources and Providers Prioritization.

Resource Registry

Firgure 10: Resource Registry Architecture

The resources registry maintains all of the root resources in the form of Resource Records.
A Resource Record holds the following:

- **URI Template Processor** - represents a URI template associated with a resource. Used during the resource matching process.
- **Resource Metadata** - holds the resource metadata collected from the resource annotations.
- **Sub-Resource Records** - records of all the sub-resources (methods and locators) collected from the sub-resource annotations.
- **Resource Factory** - a factory that retrieves an instance of the resource in accordance to the creation method defined for the resource.
  Possible creation methods include:
  - - singleton
    - prototype
    - spring configuration
    - user customizable