Web Services Linker

Redirection Notice

This page will redirect to TECHNOS:Web Services Linker.

On this page:

Target audience:

CAST Administrators

Summary: This document provides technical information about the extension called "**Web Services Linker**" (**com.castsoftware.wbslinker**).

Description

The "**Web Services Linker**" (**com.castsoftware.wbslinker**) extension is a "dependent" extension that automatically creates links for "web services" between client and server components by following a particular protocol based on objects and names. The extension can be downloaded on its own as a standalone extension, however, it is **usually automatically downloaded** as a dependency with other extensions such as:

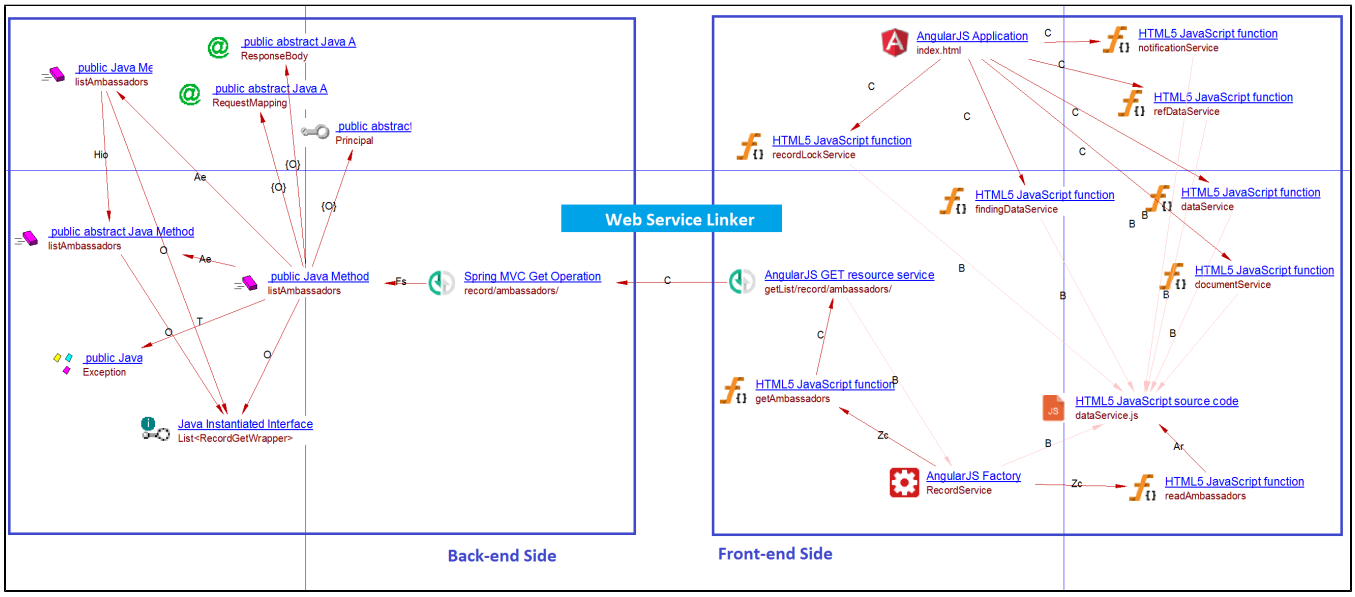
- HTML5 and JavaScript
- AngularJS
- Node.js
- jQuery
- SAPUI5
- WCF
- JAX-RS
- Spring MVC
- Objective-C Analyzer

For example for the **AngularJS** extension:

Last published	2017-09-18
Version	1.5.0-beta1 **beta**
Author	CAST Product
Dependencies	**CAIP** (>= 7.3.4) **com.castsoftware.html5** (>= 1.6.0-beta1) **com.castsoftware.wbslinker** (>= 1.3.0-beta1)

What does it do?

The Web Service Linker automatically creates **cross-technology call links** between client (front end) and server (back end) objects. For example, **AngularJS front end** connected to **JEE/Spring MVC back end** (*click to enlarge*):



How does it do it?

> End-users do not need to interact or configure the Web Services Linker extension, all configuration is automatic.

The Web Services Linker supports two modes:

- **REST services**
- **WSDL/SOAP services**

The connection is made via four "root" objects:

	HTTP **GET** Service		HTTP **DELETE** Service
	HTTP **POST/SOAP** Service		HTTP **PUT** Service

Note that you can view a list of errors and warnings that may potentially be returned during an analysis in Web Services Linker.

REST services

The Web Service Linker searches for objects stored in the **CAST Analysis Service schema** whose type inherits from **CAST_ResourceService** or **CAST_WebServiceLinker_Resource**. These objects represent queries to web services on the client side. Then it searches for the web services on the server side: these are objects whose type inherits from **CAST_WebService_Operation** or **CAST_WebServiceLinker_Operation**.

When a match is found using the properties **CAST_ResourceService.uri / CAST_WebServiceLinker_Resource.uri** and **CAST_WebService_Operation.identification.name / CAST_WebServiceLinker_Operation.identification.name** and type of both objects, then a link is created.

Matching algorithm

The matching is done between using the properties **CAST_ResourceService.uri / CAST_WebServiceLinker_Resource.uri** and **CAST_WebService_Operation.identification.name / CAST_WebServiceLinker_Operation.identification.name**.

Before matching, the Web Services Linker transforms the **CAST_ResourceService.uri** using following rules, in this order:

- It replace all "**//**" with "**/{}/**" except "**//**" after "**:**" (to avoid replacing "**http://**"), supposing that a parameter was intended between both "**/**" (REST format).
- It removes everything after "**?**" in the uri (to suppress uri parameters part which are not part of REST format parameters).
- It adds a "**/**" at the end of uri when not present

CAST_ResourceService.uri	After transformation
https://www.castsoftware.com/offices//phone//	https://www.castsoftware.com/offices/{}/phone/{}/
https://www.castsoftware.com/offices//phone/	https://www.castsoftware.com/offices/{}/phone/
https://www.castsoftware.com/offices//phone	https://www.castsoftware.com/offices/{}/phone/
https://www.castsoftware.com/offices/{}/phone	https://www.castsoftware.com/offices/{}/phone/
https://www.castsoftware.com/offices?office=1	https://www.castsoftware.com/offices/

The result is then compared to **CAST_WebService_Operation.identification.name** / **CAST_WebServiceLinker_Operation.identification.name** using the **endswith function**, ignoring the uri part corresponding to the operation name part whose value is **{}**.

CAST_ResourceService.uri	CAST_WebService_Operation.identification.name	Match: Yes/No
.../path1/path2/path3/	path4/path3/	No
.../path1/path2/path3/{}/	path3/{}/	Yes

.../path1/path2/path3/	path2/path3/	Yes
.../param1/value1/param2/value2/	.../param1/{}/param2/{}/	Yes

Examples of matches

A client side url like:

- "**https://maps.yahoo.com/place//?addr=Meudon%2C%20Ile-de-France%2C%20France**"

is transformed as:

- "**https://maps.yahoo.com/place/{}/**".

It will match a server side operation whose name is "**place/{}/**"

Examples

Server side

The Analysis Service schema contains an object **CAST_WebService_GetOperation** named **/users/**

```
@RequestMapping("/users")
public class UserController {

    @RequestMapping(method = RequestMethod.GET, produces = "application/json; charset=utf-8")
    @ResponseBody
    public PagedResources<UserResource> collectionList(...){
           ...
    }
```

Client side

The Analysis Service schema contains an object **CAST_AngularJS_GetResourceService** whose property **CAST_ResourceService.uri** equals **'resources/scenarios/{}/'**

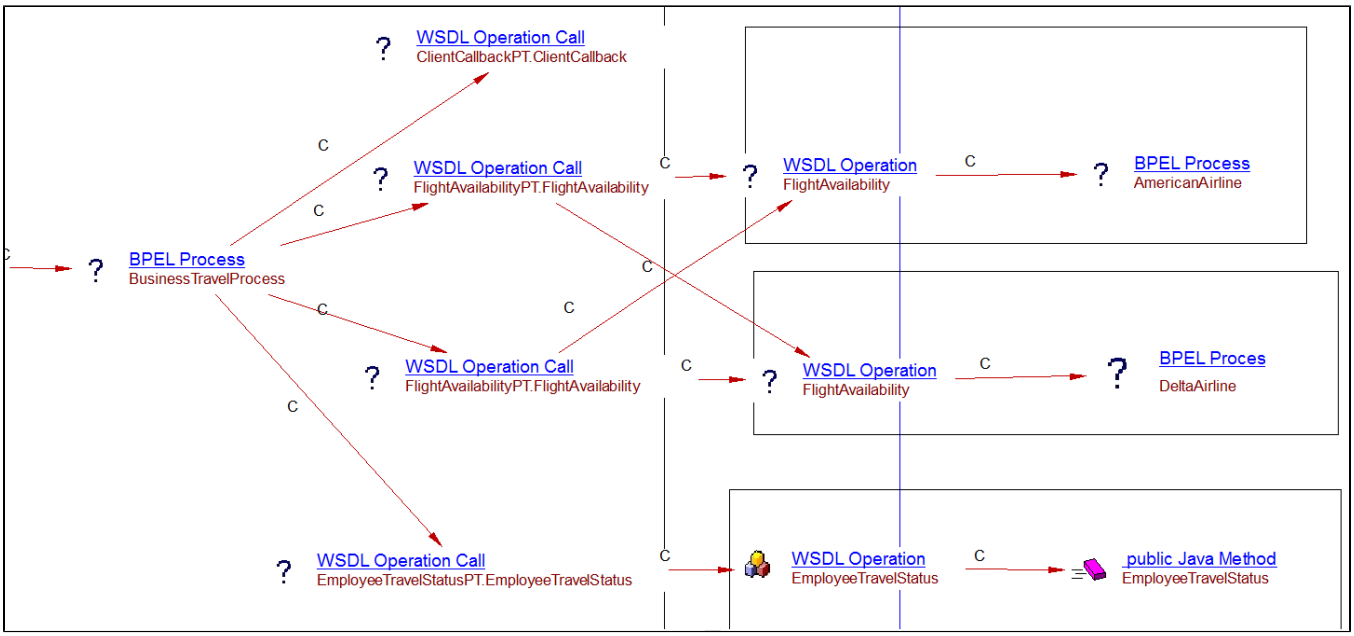
```
return $resource('resources/scenarios/:id', {'id':'@id'}, {
        'query': { method: 'GET', isArray: false },
        'save': {method:'POST', isArray: false },
        'update': {method:'PUT', isArray: false },
        'notify': {method:'PUT', params: {notify: true}, isArray: false },
        'remove': {method:'POST', isArray: false, headers:{'X-HTTP-Method-Override':'DELETE'}}
```

WSDL/SOAP services

WSDL is generally used in the context of SOAP web services: calls are to an operation, and operations are identified by :

- operation name
- port type

Samples



Client side

BPEL

Invocation of **EmployeeTravelStatusPT.EmployeeTravelStatus**:

```
<invoke partnerLink="employeeTravelStatus"
        portType="emp:EmployeeTravelStatusPT"
        operation="EmployeeTravelStatus"
        inputVariable="EmployeeTravelStatusRequest"
        outputVariable="EmployeeTravelStatusResponse" />
```

Server side

JAX-WS

Reception of operation **EmployeeTravelStatusPT.EmployeeTravelStatus**:

```
import javax.jws.WebService;

@WebService(targetNamespace = "http://superbiz.org/wsdl")
public class EmployeeTravelStatusPT {

    public int EmployeeTravelStatus(int add1, int add2)
    {

    }
}
```

BPEL

Reception of operation TravelApprovalPT.TravelApproval:

```
<receive partnerLink="client"
        portType="trv:TravelApprovalPT"
        operation="TravelApproval"
        variable="TravelRequest"
        createInstance="yes" />
```

Cross-Technology Transaction

Front-End/Service Exit Point	Back-end/Service Entry Point

• HTML5/Javascript
  • Web Socket Service (WebSocket)
  • XMLHttpRequest Service (XMLHttpRequest)
  • Http Request Service (HttpRequest,Fetch, Axios, SuprAgent)
• AngularJS
  • AngularJS Service ($resource)
  • Restangular Service (Restangular)
  • Http Service ($http)
• jQuery
  • jQuery Service ($.ajax, $.get, $.getJSON)
• iOS
  • NSURLConnection, NSURLSession
• ASP.NET
  • SOAP Resource (SoapDocumentAttr, WebMethodAttr)
  • Razor HttpRequest
• Python
  • Urllib, Urllib2, Httplib, Httplib2, aiohttp, Flask
	• Node.js
	  • Express Service (Express)
	  • Http Service (Http)
	• JEE
	  • JAX-RS (@GET, @POST)
	  • SpringMVC (@Request/Put/Post/Get/Mapping)
	• .NET
	  • WCF Operation (OperationMethod)
	• ASP.NET
	  • SOAP Operation (WebServiceAttribute)
	• Python
	  • aiohttp, Flask