Intersection Engine - Users Guide

Contents

Introduction

Overview

The GeoSpatial Intersection Engine (aka Dynamic Geographic Query Engine) is a tool for generating spatial intersections across geographical layers.

Background

- See the original Use Case UML for Dynamic Geographic Queries on the FI Spatial Infrastructure
- Check out the related Use Cases for dynamic geographic queries

Technical Requirements

1. Use a different web framework. The current implementation relies on Wicket and Ext. Wicket generates URLs which are not compatible with the FAO web servers security filters. Moreover, it is not considered the recommended framework within FIGIS. See the Web Framework analysis results.
2. Avoid relying on complex ESB and Messaging Systems solutions. ServiceMixand Camelare great enterprise integration platforms but not really in the scope of this exercise.
3. Avoid overriding intersection results. The result of an intersection should be maintained until the same intersection is requested to be recalculated, it should not be overwritten by the next intersection calculation batch.

Functional Requirements

Base Layers

- update to Geoserver 2.1 for equal area projection support (Eckert IV)
- need to provide FI Geoserver with layers well geo-coded:
- For instance:
  e.g. 1: FAO_MAJOR_AREA: well geo-coded with a single code F_AREA for each geo-feature
  e.g. 2: SPECIES_DIST: not well geo-coded: the geocode is a simple integer field.
- provide single geo-features by unique combination of attributes (in this case: ALPHACODE, DISPORDER, PRESENCE).
  - to improve the geo-coding system (here for SPECIES DIST an integer may not be appropriate), would based on a combination (concatenation) of the attributes involved in defining the geo-features. On the contrary: a multiple attributes selection may be possible instead of a unique SRC_CODE (see Geospatial Query Engine p. 10, 14) but it would imply a modification of the STATISTICAL TABLE SCHEMA.

Algorithm / Intersection

- Add projection process in the algorithm using an Equal Area projection (for area calculation purpose), since layers stored in Geoserver are in WGS84 CRS.

The alternative would be to project all layers in Geoserver in an global equal area projection, in this case, checks regarding on the fly reprojections would be necessary on map viewer applications such as the RFB map viewer.

- A Difference/Erase process would apply similarly as in the current algorithm, to “remove” eventual land overlay
- Accurate Area calculation (based on Equal Area projection)
- Intersection process:
  - multiple attributes selection (multiple columns) if the source layer geo-code is not explicit
  - multiple attribute values selection (e.g. for FAO Major areas : a list of 27,47,31 fao major areas)
- Guarantee the uniqueness of each combination: Each attributes relationship may correspond to a unique geo-feature. This may implies using an additional geometry Union process.
- Discuss the terminology used until now regarding SOURCE and TARGET: An intersection process does not have any order in selecting the base layers. The use of SOURCE and TARGET definitions may not be appropriated and may be confusing since both layers used in the intersection are SOURCE layers. Using a layer 1 as source and a layer 2 as target, or the contrary, would lead to the same combination result.
- Precisions are needed regarding:
  - the database storage of intersection geometries (and possible errors as mentioned in the technical documentation)
  - the support (or not) of layers in which polygons are not exclusive and can overlap between each others (topology not respected).
  - the possibility for the Download manager to support other output Formats than CSV, XML, GML2

Use Cases

1. Feed the water area intersection table
2. Calculate the intersection between two (layer, area) tuples via (REST) web service
3. Scenarios for map viewers:
   a. Species Map Viewer: end-user request one or more species distribution areas and the application returns a list of all interstecting layers (i.e., RFBs, EEZs, High-seas, LMEs, Country (in case of Inland Species), FAO areas) and the intersection area percentage.
   b. Regional Fisheries Bodies Map Viewer: end-user request one RFB area of competence and the application returns a list of all interstecting layers (i.e., species, other RFBs, EEZs, High-seas, LMEs, Country (in case of Inland RFB), FAO areas) and the intersection area percentage.
   c. Each intersecting layer is displayed in the map at user request and the result of the computed intersections is exportable in printable or re-usable format (e.g., PDF, CSV).

[09 June 2011 - 16:48] Meeting Report and proposed solution for first phase

NOTES: It is worth to point out that the report here below is still a draft since some requisites, especially on client side, need to be better defined. Also the proposed solution envisages a first release of Intersection Engine which aimed to be the base for further development as new requirements will be arised or the old one will be changed somehow or better specified.

Examining in details the requirements and use cases with Francesco, Fabio, Aureliano and Alessio, we established for this first phase as follows:

1. The Intersection Engine will get as input an XML conifguration file listing all the possible couple of Sources and Targets. Each line should contain the following information:

   SRC_SYSTEM / TARGET_SYSTEM / SRC_CODE_FIELD / TARGET_CODE_FIELD / PRESERVE_SRC_GEOM_OR_NOT

   e.g.: SPECIES / FAO_DIV / ALPHACODE / F_AREA / FALSE

   the user can specify many configuration. The IE won't take into account equal rows anyway.

2. For each configuration will be generated two different intersecting tables:
   - STATISTIC_TABLE [IE_SRC_TRG_SRCCODE_TRGCODE] --> feature_id / src_system / trg_system / src_code_field_name / src_code_field_value / trg_code_field_name / trg_code_field_value / src_area / trg_area / src_overaly_percentage / trg_overlay_percentage

   - GEOMETRY_TABLE [IE_SRC_TRG_SRCCODE_TRGCODE_GEOM] --> feature_id / geometry

NOTE: *we are still evaluating if it would be better having only one table for all statistics or create a new one for each couple.*

3. We envisage the use of complex code_fields for layers in order to better geo-code the outcomes. For instance another SPECIES_EXT layer will be created containing a code_field defined as the unione of 3ALPHA_PRESENCE_DISPORDER

   e.g.: ALB_0_0 can be a possible value of this "composite" field

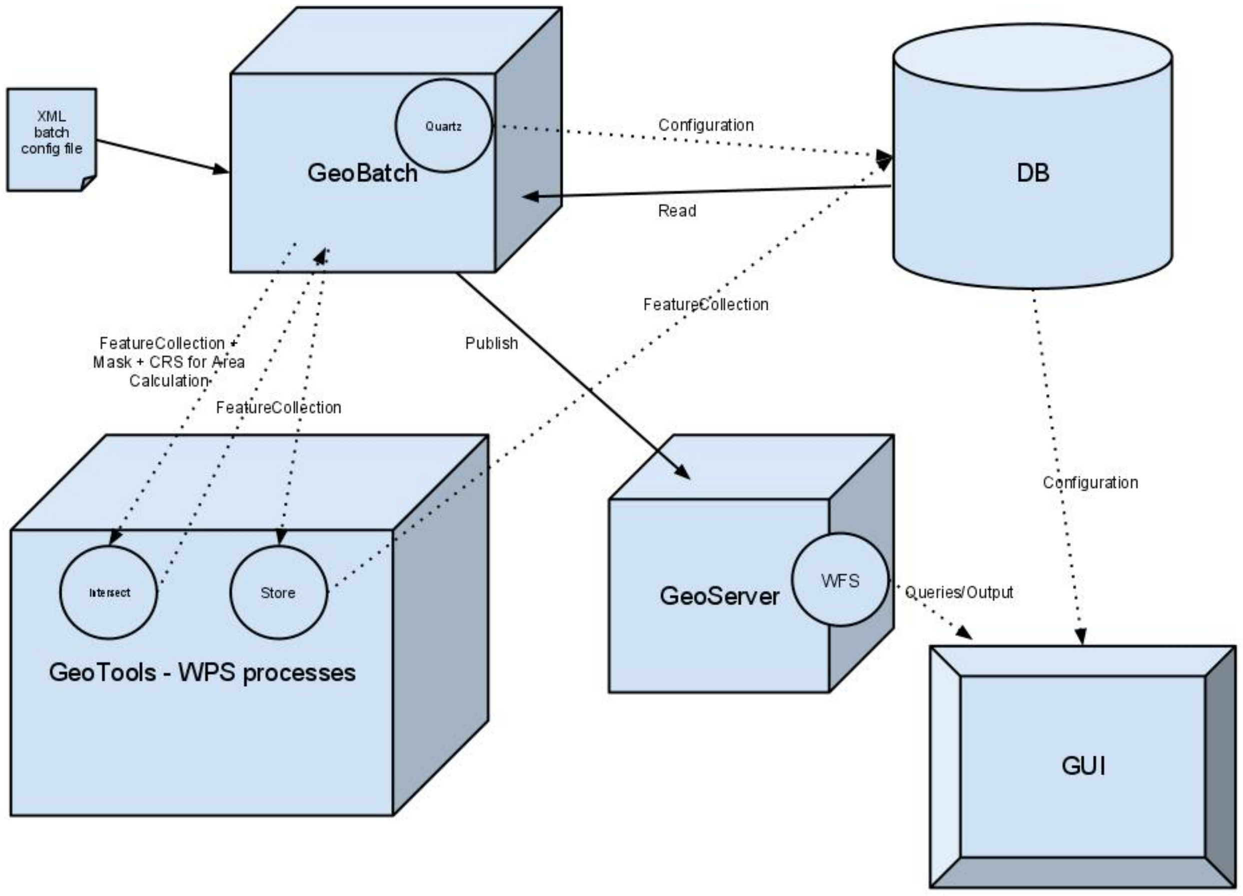
4. A cron task will launch periodically an Intersection Engine trigger which will build the intersecting tables and layers on GeoServer following the specified algorythm.

5. The Intersection Engine will allow users to invalidate generated intersecting tables which will be rebuilt. Otherwise the outcomes will be maintained by the IE trigger and won't be generated again.

6. The outputs will be provided by the WFS in different output formats: XML, CSV, GML2, RDF, SHP ... others may be defined later.

Intersection Engine Architecture

Blocks-schema and Overview



The above diagram depicts a blocks-schema of the whole Intersection Engine architecture.
The central core of the Intersection Engine business logic is provided by GeoBatch, a pluggable web application able to execute complex flows as a chain of a set of simple actions.
Each GeoBatch action is delegated to manage a part of the whole workflow (see details below into the “Intersection Engine Flow” and “Intersection Engine: GeoBatch Actions” sections).
Moreover the GeoBatch logic makes use of GeoTools and GeoServerManager libraries in order to respectively execute geometric processes of the flow and pubblication of the results on GeoServer.

At an high level view of the whole architecture, the GeoBatch checks for Intersection Engine configuration updates (those can be handled through XML files as described in details on the next sections). During this phase integrity checks are made in order to maintain an always coherent configuration with the actual status of the computed intersections.

Once the intersections status has been updated, GeoBatch prepares the instructions to pass to the Intersection Action which has the responsability of both computing the intersections and cleaning the wrong or old ones too.
As the new intesections becomes ready they are published or updated on GeoServer.
Every client will be able to access the results through the GeoServer web services. The Intersection Engine GUI also will expose the results by bridging calls to GeoServer OGC web services.

In the next section will be described the above workflow in more details and presented the real components composing the blocks.

Intersection Engine GeoBatch Flow details

Algorithm and workflow

Algorithm Description

1. The algorithm takes as input two tables with a spatial geometry of type Multi-Polygon
2. The algorithm computes the percentages of the intersections of the source layer with the second one. The quantitative percentages have to be stored on a separate statistical table which does not have any kind of spatial column, while on another table will be stored the resulting intersection polygons. The keys must match of course, and the computation must take in consideration the COASTLINE polygons as mask taken from an other GIS layer outside.

Note

The resulting tables must be the same for each computation, a complex key reporting the source layer and the target layer IDs will be used to differentiate the computation records.
In case a computation is issued again, the old related records must be removed from the table.
Other than the final tables must be always consistent, so in order to do that the outcomes must be stored on a temporary table and copied when the computation finishes.
The source layer and the target layer have to be both masked with the land layer before the computation, in order to take care only of the OCEAN areas.
The resulting intersections have to be the cartesian product between the source layer polygons and the target layer polygons, basically we have to consider ALL the possible intersections among the target and source layer. As an example if 1 polygon of the source layer intersects 3 polygons of the target layer, we will have 3 records on the resulting statistical table.

STATISTICAL TABLE SCHEMA

Column	Description
ID	Automatically generated
SRC_NAME	Acronym of the source layer
SRC_CODE	Unique Code of the source layer (provided by external param)
TOT_AREA_SRC	Total surface in sqm of the whole src layer polygon
TRG_NAME	Acronym of the target layer
TRG_CODE	Unique Code of the target layer (provided by external param)
TOT_AREA_TRG	Total surface in sqm of the trg layer polygon
AREA	Surface in sqm of the resulting intersection polygon
OV_SRC	Single polygon vs area of source (AREA/TOT_AREA_SRC) * 100
OV_TRG	Single polygon vs area of target (AREA/TOT_AREA_TRG) * 100

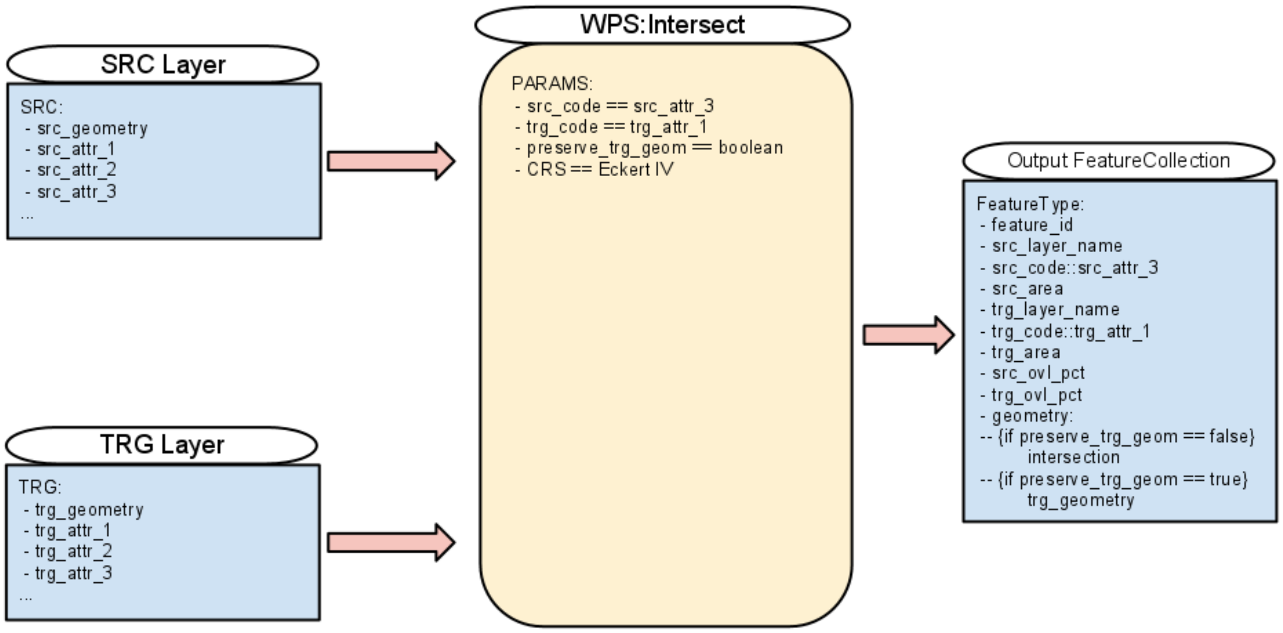
SPATIAL TABLE SCHEMA

Column	Description
ID	Automatically generated
STATISTIC	Foreign Key to statistics table
THE_GEOM	Resulting Geometry

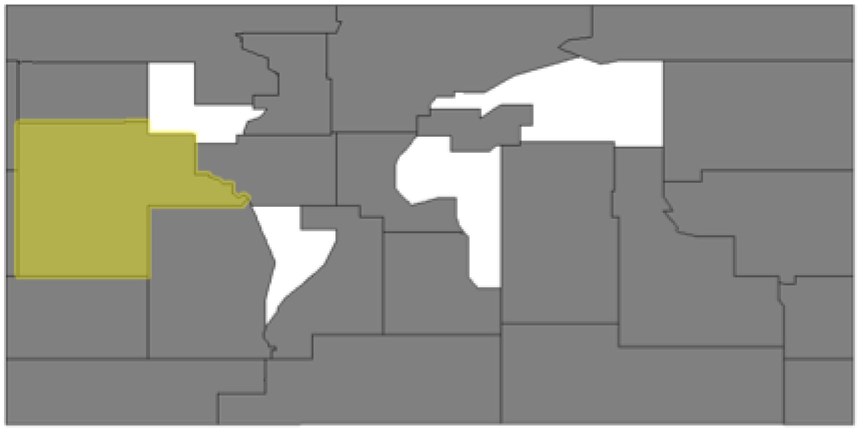
Example

```
src->FAO_MAJOR; trg->NJA
ID: 1
SRC_NAME: FAO_MAJOR
SRC_CODE: 47
TOT_AREA_SRC: 1871,4226412174348
TRG_NAME: NJA
TRG_CODE: -
TOT_AREA_TRG: 37,575021697177185
AREA: 37,575021697177185
OV_SRC: 2,0078319493202850278361752966668
OV_TRG: 100,00


ID: 2
SRC_NAME: FAO_MAJOR
SRC_CODE: 47
TOT_AREA_SRC: 1871,4226412174348
TRG_NAME: NJA
TRG_CODE: -
TOT_AREA_TRG: 78,3211774694
AREA: 78,3211774694
OV_SRC: 4,18511434800473299326961625797744
OV_TRG: 100,00
```

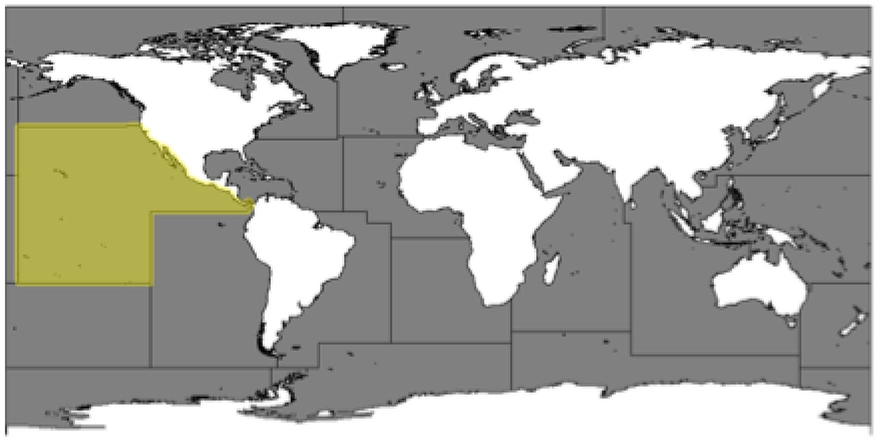

SRC Layer
SRC:
- src_geometry
- src_attr_1
- src_attr_2
- src_attr_3
...
TRG Layer
TRG:
- trg_geometry
- trg_attr_1
- trg_attr_2
- trg_attr_3
...
WPS:Intersect
PARAMS:
- src_code == src_attr_3
- trg_code == trg_attr_1
- preserve_trg_geom == boolean
- CRS == Eckert IV
Output FeatureCollection
FeatureType:
- feature_id
- src_layer_name
- src_code::src_attr_3
- src_area
- trg_layer_name
- trg_code::trg_attr_1
- trg_area
- src_ovl_pct
- trg_ovl_pct
- geometry:
-- {if preserve_trg_geom == false} intersection
-- {if preserve_trg_geom == true} trg_geometry

Graphical Overview
Source: FAO_MAJOR

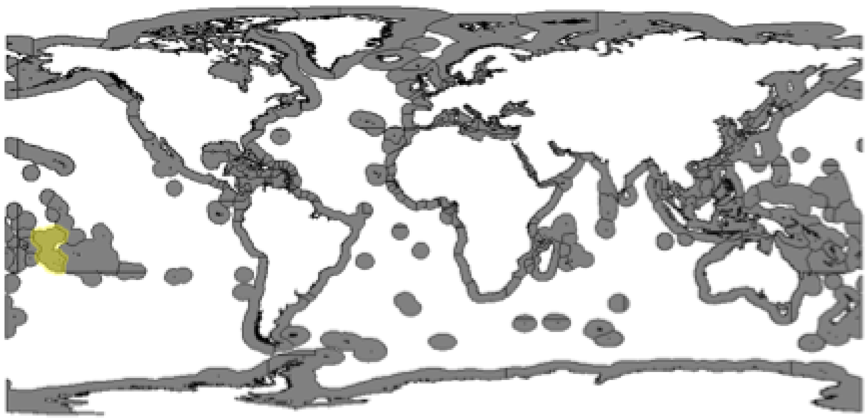


Mask: UN_CONTINENT

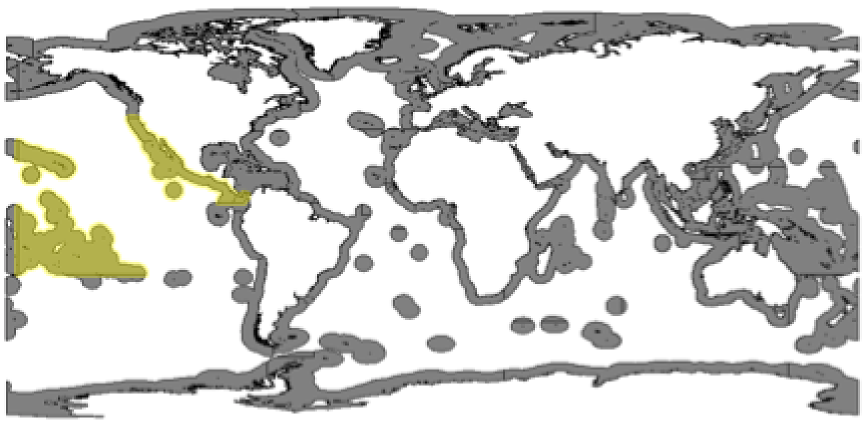
Masked Source: FAO_MAJOR minus UN_CONTINENT



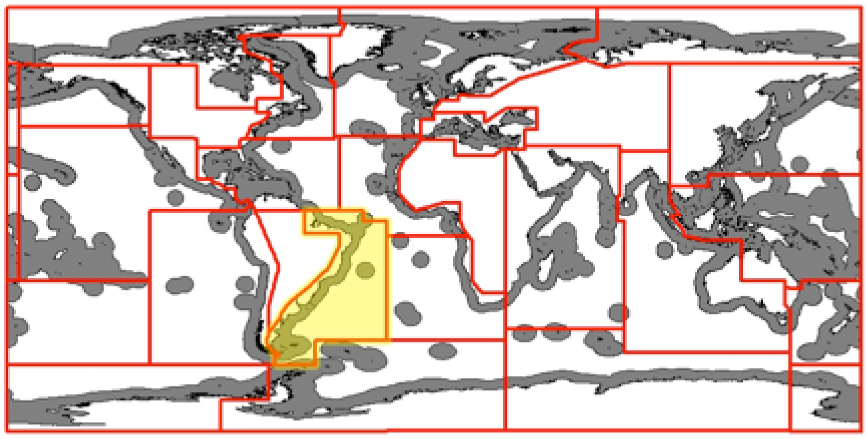
Target: NJA



Intersection

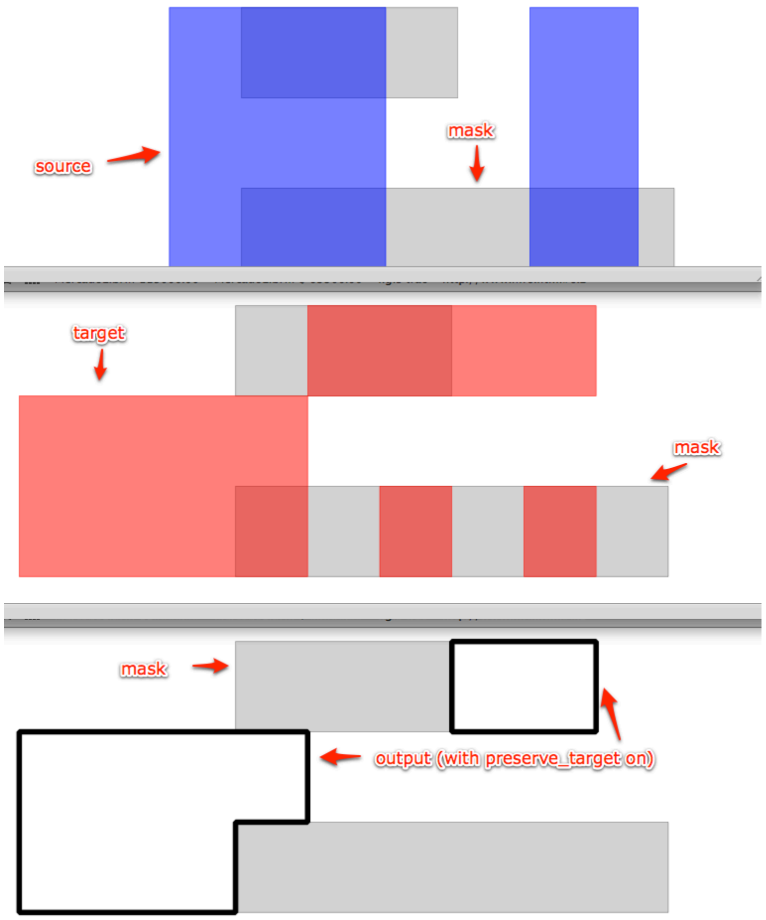


Nicer image showing final result

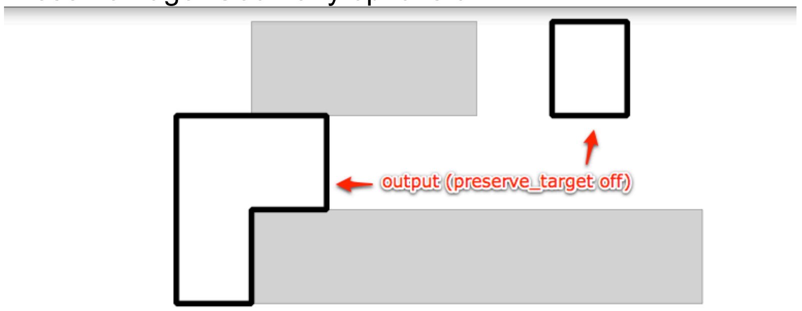


Preserve Target Geometry option

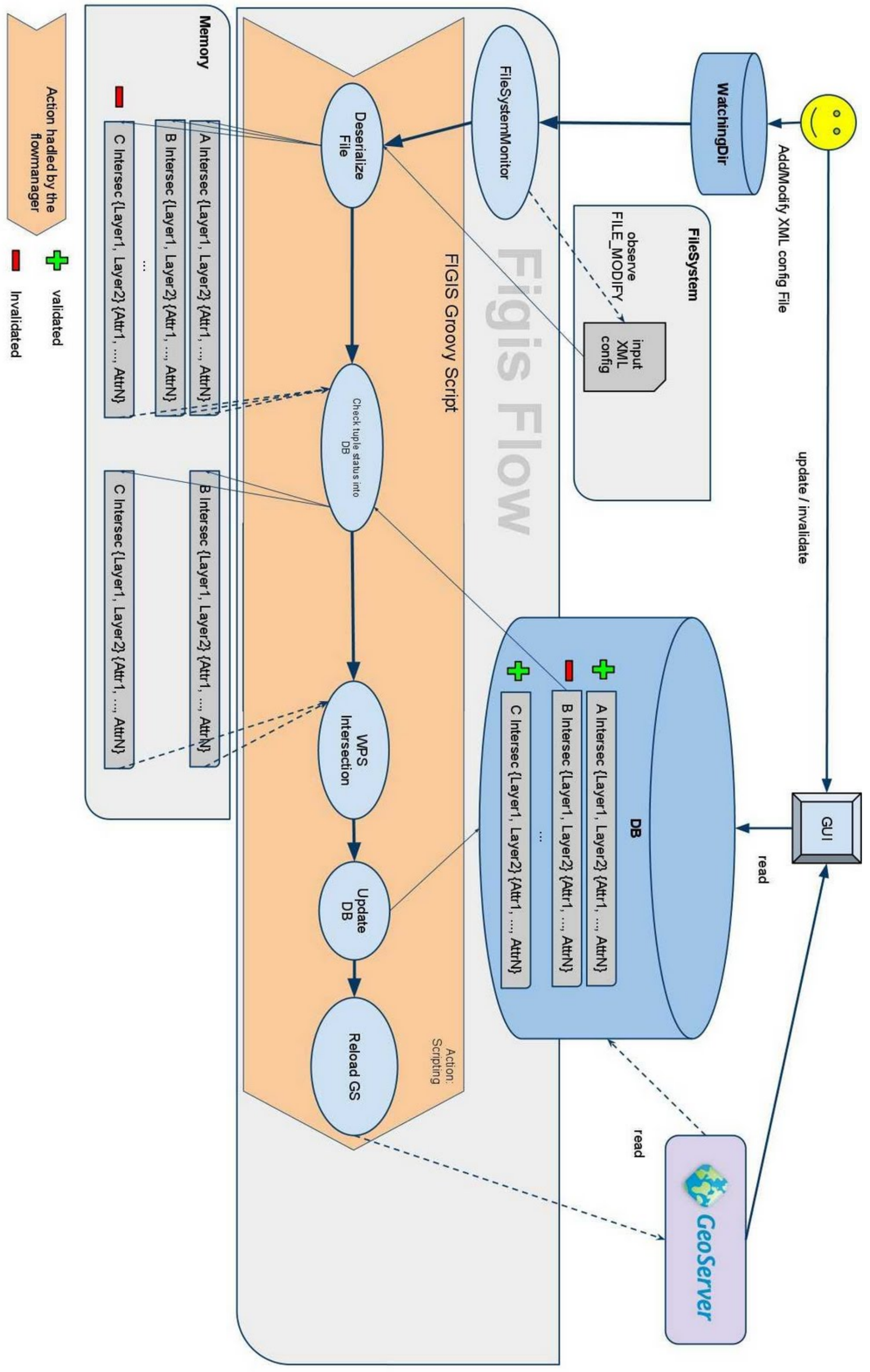
It's an optional parameter that saves the masked target polygons instead of just the sections that intersect the source.
Here below a graphical explanation:



Preserve Taget Geometry options off:



GeoBatch Workflow


Add/Modify XML config File
WatchingDir
FileSystem
observe FILE_MODIFY
input XML config
update / invalidate
Figis Flow
FileSystemMonitor
FIGIS Groovy Script
Deserialize File
Check tuple status into DB
WPS Intersection
Update DB
Reload GS
Action: Scripting
GUI
read
read
DB
A Intersec {Layer1, Layer2} {Attr1, ..., AttrN}
B Intersec {Layer1, Layer2} {Attr1, ..., AttrN}
C Intersec {Layer1, Layer2} {Attr1, ..., AttrN}
...
GeoServer
Memory
A Intersec {Layer1, Layer2} {Attr1, ..., AttrN}
B Intersec {Layer1, Layer2} {Attr1, ..., AttrN}
C Intersec {Layer1, Layer2} {Attr1, ..., AttrN}
...
B Intersec {Layer1, Layer2} {Attr1, ..., AttrN}
C Intersec {Layer1, Layer2} {Attr1, ..., AttrN}
Action hadled by the flowmanager
validated
Invalidated

The GeoBatch workflow is quite complex; lets try to examine in details the most important components of the workflow.

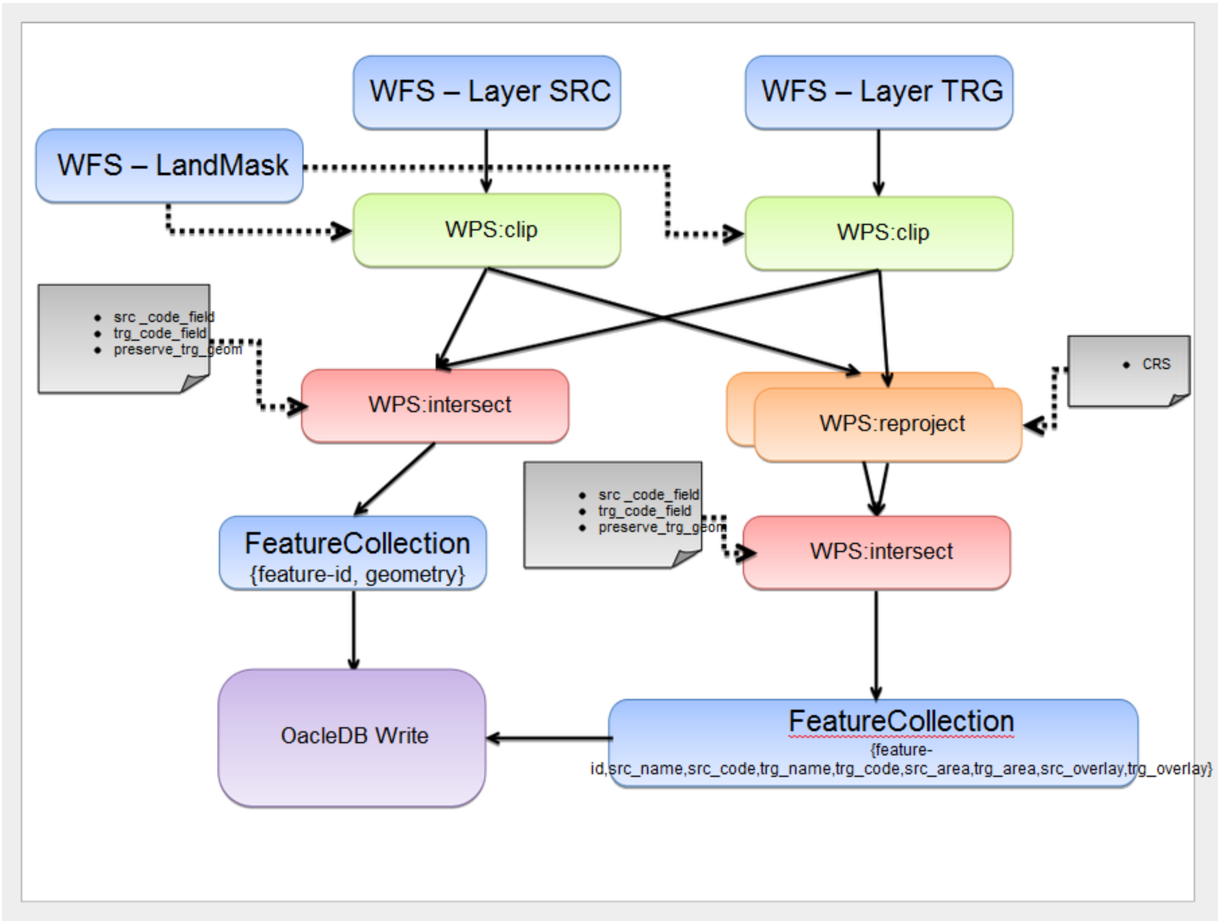
1. FileSystemMonitor: it’s a GeoBatch mechanism which allows it to recognize changes on the file system. Several events can be catched by GeoBatch which is able to know for a certain folder if something has been updated, added or removed. GeoBatch configuration also allows to discriminate unusefull from important events through specific regular expressions. In this context this mechanism is used to recognize

updates of Intersection Engine configuration, provided through XML files with increasing “updateVersion” (see sections below for details).

2. After the configuration deserialization, integrity checks are made by the system in order to establish which Intersections couples must be computed, removed or leave untouched. During this phase GeoBatch also collects input datasets and performs mask clipping if required. Then, the responsability is passed to the WPS Intersection process, which is delegated to perform the real intersection process and produce the expected outputs.
3. On a final phase the DB is updated. If old intersection are present, they will be updated only if new ones of the same kind are available and correctly computed. New one are always added if correctly processed instead.
4. At the end GeoServer is updated and the datasets made available for download through the GUI.

Intersection Engine GeoTools processes

Processes Chain Diagram



The diagram above is a graphical representation of the GeoTools Processes chain performed by GeoBatch Actions.
Starting from the top, the data is collected by downloading the FeatureCollections as zipped shapefiles from GeoServer along with the requested Land Mask.
A “Clip” process is applied to the source FeatureCollections in order to remove the land part.
A reprojection is also performed if requested in order to correctly compute the output area overalys. Usually an equal area projection is used to reproject the input FeatureCollections.
Finally the “Intersection” process takes care of performing all the intersections and produce two different outcome FeatureCollections, a pure statistical one containing informations on the source and target codes and area overlays, and a pure spatial one containing the resulting intersected geometries.
Those two FeatureCollections are then stored to the DataBase and made available to internal users which can directly access the tables or to the external ones through the GeoServer Web Services.

Intersection process

This is the detailed description of the reusable process to compute intersections and areas

Inputs: maskCollection, featureCollection1, featureCollection2, areaCRS
Output: a new feature collection whose structure is att1, att2, the intersected geometry, src area, target area, src_overlay_perc, target_ovelay_perc

The mask collection is optional. GeoBatch is supposed to get the two collections from the database directly, already shaving off all attributes but the two that are requested in its configuration.
The two input feature collections are supposed to be in the same CRS too.
The areaCRS is a CRS that will be used to compute the areas of the various polygons generated by the process

Main computation “loop” (it will be done in a feature collection iterator actually, to avoid storing everything in memory):

- get the mask bbox, the fc1 and fc2 bboxes, intersect them all -> that’s the working area
- for each feature in fc1 inside the working area
    - get the corresponding mask features, union their geometry, get the clipping geometry as a result
    - clip the geometry of fc1
    - reproject the geometry in the areaCRS and compute the area
    - get all features in fc2 that can intersect the clipped fc1 geometry
    - for each of the fc2 geometries
        - clip the fc2 geometry to the mask
        - reproject the result in areaCRS to compute the original area
        - intersect with fc1 (this is the result geometry) and clean up eventual point and line elements (they will happen if we intersects geometries that do touch in some parts without actually intersecting)
        - reproject the result in areaCRS to compute the intersected area
        - compute percentages
        - emit the resulting feature with all the non geometric attributes of the first and second feature, the intersected geometry, the src, target area and the overlap percentages

Store process

This one is FIGIS specific
Inputs: statsFeatureCollection, sourceSystem, targetSystem, oracleStore, statsTableName, geomTableName
Outputs: none, will just give back a return status (since we can’t have a process with really 0 ouptuts)

The oraclestore is a geotools JDBCDataStore that GeoBatch already used to grab the feature collections, we’ll get the connection to do the inserts from its connection pool (btw, geobatch could keep the connection pool or the store itself around).

The process will first create two temporary target tables deriving details from the feature collection (the attribute names and types) and the source and target system, then scroll through the collection and perfom batches of inserts in the db (a transaction every, say, 100-1000 records to avoid overwhelming the db, then commit and restart the transaction).
Once done it will remove the eventual old tables and rename the temp ones to their definitive names (in a single transaction)

It is the responsibiliy of GeoBatch to configure the resulting sql view in GeoServer for publishing purposes.

Installation and Configuration

Prerequisites

Java 5+

Because of the use of annotations and some other features Java 5 is the minimum version supported by IE. Java 6 is of course recommended for performance reasons.

Maven 2+

The project takes advantage of Maven for package management and building. It has been developed with version 2.0.9 but some earlier and further versions should work just fine.

Spatial Database (Oracle 10g+)

Oracle 10g has been used for development, but any spatial database supported by Geoserver 2.1 and above will work. If you use Oracle, please remember to use the Oracle NG drivers.

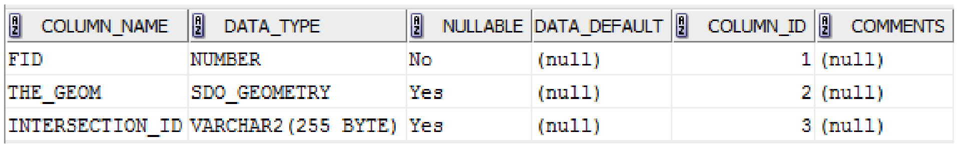
First startup and DB initialization

When the application starts for the first time, it creates automatically 4 tables on the target DB:

- SPATIAL_TABLE
- SPATIAL_TMP_TABLE
- STATISTICAL_TABLE
- STATISTICAL_TMP_TABLE

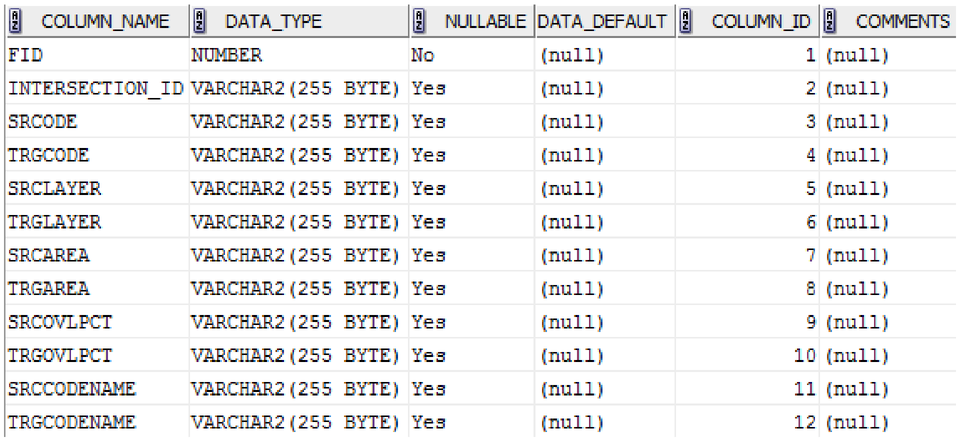
defined as follows:

SPATIAL_TABLE / SPATIAL_TMP_TABLE



COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
FID	NUMBER	No	(null)	1	(null)
THE_GEOM	SDO_GEOMETRY	Yes	(null)	2	(null)
INTERSECTION_ID	VARCHAR2(255 BYTE)	Yes	(null)	3	(null)

SPATIAL_TABLE / SPATIAL_TMP_TABLE



COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
FID	NUMBER	No	(null)	1	(null)
INTERSECTION_ID	VARCHAR2(255 BYTE)	Yes	(null)	2	(null)
SRCODE	VARCHAR2(255 BYTE)	Yes	(null)	3	(null)
TRGCODE	VARCHAR2(255 BYTE)	Yes	(null)	4	(null)
SRCLAYER	VARCHAR2(255 BYTE)	Yes	(null)	5	(null)
TRGLAYER	VARCHAR2(255 BYTE)	Yes	(null)	6	(null)
SRCAREA	VARCHAR2(255 BYTE)	Yes	(null)	7	(null)
TRGAREA	VARCHAR2(255 BYTE)	Yes	(null)	8	(null)
SRCOVLPCT	VARCHAR2(255 BYTE)	Yes	(null)	9	(null)
TRGOVLPCT	VARCHAR2(255 BYTE)	Yes	(null)	10	(null)
SRCCODENAME	VARCHAR2(255 BYTE)	Yes	(null)	11	(null)
TRGCODENAME	VARCHAR2(255 BYTE)	Yes	(null)	12	(null)

Few preliminary operation have to be performed the first time in order to ensure the correct functionality of the spatial DB though:

1. Clean up spatial metadata and indexes

```
DELETE FROM MDSYS.SDO_GEOM_METADATA_TABLE WHERE sdo_table_name=upper('SPATIAL_TABLE');
COMMIT;

DELETE FROM MDSYS.SDO_GEOM_METADATA_TABLE WHERE sdo_table_name=upper('SPATIAL_TMP_TABLE');
COMMIT;

DROP INDEX "FIGIS_GIS"."SPATIAL_TMP_TABLE_THE_GEOM_IDX";
COMMIT;

DROP INDEX "FIGIS_GIS"."SPATIAL_TABLE_THE_GEOM_IDX";
COMMIT;
```

2. Add spatial metadata

```
INSERT INTO MDSYS.SDO_GEOM_METADATA_TABLE VALUES (
     'FIGIS_GIS',
     'SPATIAL_TABLE',
     'THE_GEOM',
     MDSYS.SDO_DIM_ARRAY(
         MDSYS.SDO_DIM_ELEMENT('X',-180,180,0.005),
         MDSYS.SDO_DIM_ELEMENT('Y',-90,90,0.005)
     ),
     4326
  );
 COMMIT;

INSERT INTO MDSYS.SDO_GEOM_METADATA_TABLE VALUES (
     'FIGIS_GIS',
     'SPATIAL_TMP_TABLE',
     'THE_GEOM',
     MDSYS.SDO_DIM_ARRAY(
         MDSYS.SDO_DIM_ELEMENT('X',-180,180,0.005),
         MDSYS.SDO_DIM_ELEMENT('Y',-90,90,0.005)
     ),
     4326
  );
 COMMIT;
```

3. Create spatial indexes in the correct target CRS

```
 CREATE INDEX "FIGIS_GIS"."SPATIAL_TMP_TABLE_THE_GEOM_IDX" ON "FIGIS_GIS"."SPATIAL_TMP_TABLE"
("THE_GEOM") INDEXTYPE IS "MDSYS"."SPATIAL_INDEX" ;
 COMMIT;
```

```
CREATE INDEX "FIGIS_GIS"."SPATIAL_TABLE_THE_GEOM_IDX" ON "FIGIS_GIS"."SPATIAL_TABLE" ("THE_GEOM")
INDEXTYPE IS "MDSYS"."SPATIAL_INDEX" ;
 COMMIT;
```

Done.

GeoTools 8

GeoTools 8 is being used because of its maturity on the support of WFS, WPS and OracleDatastore (and JDBC stuff in general).

GeoServer 2.1.2+

In order to guarantee the correct functionality of Intersection Engine publishing on the outside, GeoServer 2.1.2 or above is required.

GeoServer layer initial setup

Once the tables are available on the DataBase, GeoServer needs to be configured in order to expose statistical and spatial layers to the client.

`fifao:statistical`
mapping to FIGIS.STATISTICAL_TABLE; the CRS must be set to EPSG:4326 and BBOX to (-180,-90,180,90)

`fifao:spatial`
mapping to FIGIS.SPATIAL_TABLE; the CRS must be set to EPSG:4326 and BBOX to (-180,-90,180,90)

`fifao:spatialstats`
must be an SQL View layer type (see http://docs.geoserver.org/latest/en/user/data/sqlview.html for more details) the CRS must be set to EPSG:4326 and BBOX to (-180,-90,180,90)

The SQL View must be defined as follows:
SELECT STATISTICAL_TABLE.*,SPATIAL_TABLE.THE_GEOM FROM STATISTICAL_TABLE INNER JOIN SPATIAL_TABLE ON (STATISTICAL_TABLE.INTERSECTION_ID = SPATIAL_TABLE.INTERSECTION_ID)

Building the project

The project can be built using Maven by running the following command from ie-sdk folder:

```
ie-sdk >
   mvn clean install -Dmaven.test.skip -Pdao.xstream,intersection,setting
```

ZIP archives will be created into “*distribution*” containing the two web application to be deployed on the Web Application Container.

```
gb-application-figis-2.0-SNAPSHOT.war -> rename to: geobatch.war

ie-services-2.0-SNAPSHOT.war -> rename to: ie-services.war
```

System Configuration

The distribution archive contains two folders other than the WARs to be deployed into the Web Application Container.

GEOBATCH_DATA_DIR
Contains the configuration of the GeoBatch actions and flows. It must be accessible from the Web Container which also MUST have write rights on this and all its subfolders.

In order to allow the Web Container to link this folder, an option MUST be specified to the JVM at startup:

```
GEOBATCH_DATA_DIR=/apps/apache-tomcat-6.0.16/temp/GEOBATCH_DATA_DIR
JAVA_OPTS=$JAVA_OPTS -DGEOBATCH_DATA_DIR=$GEOBATCH_DATA_DIR
```

What this folder contains …

1. GeoBatch Actions input directories (where the configuration files must be thrown in order to let the Intersection Engine compute the new intersections.
2. XML files which configure the GeoBatch Actions:
    a. **setting.xml**: configures the first action, the ones checking for new ie-config.xml inputs and performing preliminary integrity checks.
    The only parameters to tweak are:
        i. **interval**: the polling interval for the control of Filesystem changes
        ii. **ieServiceUsername** and **ieServicePassword**: username and password of ie-services REST configuration service
        iii. **persistencyHost**: base HOST where the ie-services has been deployed and running
    b. **intersection.xml**: same as above. This action does not check for Filesystem updates, just runs process at regular time intervals. Since intersection processes are heavy, it's recommended to set the interval at least at half day distance or more. Usually the Intersection Engine does not require to compute the intersections often since the data is quite static.

Final Notes:
The intersections configuration files must be put into "GEOBATCH_DATA_DIR\setting\in" in order to order new computations to the Intersection Engine.
This can be done by simply copying the new XML input file to the Filesystem location or via FTP.
GeoBatch can be also configured to start an internal FTP server pointing directly to the input folders.

HSQLDB
Contains the HSQLDB data for the ie-services REST config service.
This service stores the updates and history of changes to the configuration and instructs the Intersection Process on the computation to build or remove/cleanup.

It can be places on any point of the Filesystem and MUST be writable by the Web Application Container process.

It's location must be configured on all ***"ieServicesHsqldb.properties"*** files which can be found under **WEB-INF/classes** folders of both ie-services.war and geobatch.war.

```
### HSQL
ie-services-hsqldb.serverDatabase=file:/apps/temp/HSQLDB/data/intersectionengine
ie-services-hsqldb.serverDbname=intersectionengine
ie-services-hsqldb.serverPort=9001

ie-services-hsqldb.dataSource.driverClassName=org.hsqldb.jdbcDriver
ie-services-hsqldb.vendorAdapter.databasePlatform=org.hibernate.dialect.HSQLDialect
ie-services-hsqldb.dataSource.url=jdbc:hsqldb:hsql://localhost:9001/intersectionengine
ie-services-hsqldb.dataSource.username=sa
ie-services-hsqldb.dataSource.password=
```

The highlighted strings, represents the basic configuration of the HSQLDB database.

Moreover the **ie-services** REST service needs some Users to be configured through the **ie-services/WEB-INF/userac.properties** file

Notice that there’s no need to restart the application to updated the usernames and passwords, since the application is configured in order to automatically recognize runtime changes to this file.
The users configuration is in the form usersRoleAdmin=username@password
Example: `usersRoleAdmin=admin@abramisbrama`

IE Config.XML

In this section is described the structure of an INPUT file to be provided to the GeoBatch SETTING Action in order to issue new computation to the Intersection Engine.

A typical example of a config.xml file is the following one:

```
<?xml version="1.0" encoding="UTF-8"?>
<ie-config updateVersion="4">

   <global>
       <geoserver>
           <geoserverUrl>http://192.168.1.106:8484/figis/geoserver</geoserverUrl>
           <geoserverUsername>admin</geoserverUsername>
           <geoserverPassword>abramisbrama</geoserverPassword>
       </geoserver>
       <db>
           <database>orcl</database>
           <schema>FIGIS_GIS</schema>
           <user>FIGIS_GIS</user>
           <password>FIGIS</password>
           <port>1521</port>
           <host>192.168.1.106</host>
       </db>
       <clean>false</clean>
   </global>

   <intersections>
               <intersection mask="false" force="true" preserveTrgGeom="false">
                   <srcLayer>fifao:FAO_DIV</srcLayer>
                   <trgLayer>fifao:NJA</trgLayer>
                    <maskLayer>fifao:UN_CONTINENT</maskLayer>
                   <srcCodeField>F_SUBAREA</srcCodeField>
                   <trgCodeField>ISO3_TERRI</trgCodeField>
                   <areaCRS>EPSG:54012</areaCRS>
               </intersection>
               <intersection mask="true" force="false" preserveTrgGeom="true">
                   <srcLayer>fifao:FAO_DIV</srcLayer>
                   <trgLayer>fifao:NJA</trgLayer>
                    <maskLayer>fifao:UN_CONTINENT</maskLayer>
                   <srcCodeField>F_DIVISION</srcCodeField>
                   <trgCodeField>ISO3_TERRI</trgCodeField>
                   <areaCRS>EPSG:54012</areaCRS>
               </intersection>

               …

   </intersections>

</ie-config>
```

The configuration file is composed by two main sections **global** and **intersections**.
The first one is used to configure the access to the GeoServer instance containing the source layers and the Database where the computations must be stored.

updateVersion: *is the version of the config file. Files with an updateVersion equal or less to the actual one stored on the ie-services REST config service are completely ignored by the system.*

global / geoserver: *contains the Host URL, username and password of a user able to read from GeoServer Web and REST services. This GeoServer MUST contain all the layers specified on the intersections below.*

global / database: *contains all the access parameters to the Database where the intersections will be stored.*

Note: the password will be encrypted by the system internally so that they won't be visible by users accessing directly somehow to the ie-services REST service.

global / clean: Boolean *forcing the system to re-compute all the Intersections blindly. This parameter overrides the "force=true" for each single Intersection.*

intersections: *List of intersection couples to compute.*

intersection / mask: Boolean; *says to the system if land masking must be applied or not to the input layers.*

intersection / force: Boolean; FALSE *if an intersection with the same parameters already exists and has been correctly computed, will be preserved and this process ignored.* TRUE *re-compute the intersection and if the results is correct overwrite the old one.*

intersection / preserveTrgGeom: Boolean; *says to the system if the target layer geometry must be preserved or not. If not the final result will contain the intersections geometries between the source and target layers ones.*

srcLayer: *full qualified name of the source layer on GeoServer.*

trgLayer: *full qualified name of the target layer on GeoServer.*

maskLayer: *full qualified name of the land mask layer on GeoServer.*

srcCodeField and **trgCodeField**: *attribute name to be used as discriminator for final results values.*

Note: The above fileds represent also the **KEY** for a computation.

areaCRS: *the EPSG code of the CRS to be used to compute the area values. The input geometries will be reprojected to this one before computing the values. It SHOULD be an Equal Area CRS.*