Coverage Subsetting Analysis

# WCS Specification (OGC 09-110r4)

*https://portal.ogc.org/files/09-110r4*

*Section 8.4.2.3. Single Dimension Trimming*

**id** be the coverage identifier specified in the GetCoverage request;

**dname** be the dimension name specified in the trim request parameter;

**tLow** and **tHigh** be the trimLow and trimHigh parameter, resp., in the request, if provided

**c** be the OfferedCoverage of the server addressed;

**low** = tLow if specified in the request, otherwise low is set to the coverage’s lower bound in dimension dname;

**high** = tHigh if specified in the request, otherwise high is set to the coverage’s upper bound in dimension dname;

**B** be an envelope equal to the domain of c, except that in dimension dname the extent is given by the closed interval [low,high];

**Requirement 38 /req/core/getCoverage-response-trimming:**

The response to a successful *GetCoverage* request on coverage identifier **id** of admissible type containing no slicing and exactly one trimming operation with dimension name **dname**, lower bound parameter evaluating to **low**, and upper bound parameter evaluating to **high** shall be a coverage identical to **c**, but containing all points of **c** with location inside **B**, and no other points.

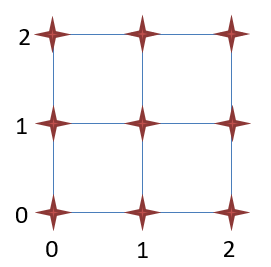
*NOTE This requirement does not specify the actual extent of the coverage returned. Possible options include: the minimal bounding box of the coverage returned, or the request bounding box. Servers are strongly encouraged to deliver the minimal bounding box.*

KS: Note seems contradictory to Req 38. How can a bbox be provided if req 38 states “and no other points”. Formally, only Requirements are normative, Notes only Informative, so Requirement beats the Note.

# FAIRiCUBE Understanding

Taking into account the definitions above, B defined by the closed interval [low,high], working on the assumption of a trivialized grid:

* Origin: 0,0
* Resolution: 1



We would expect the following behavior:

0 < low <=1; 1 <= high <2 🡪 returns 1 value from position 1

low = 1; high = 1 🡪 returns 1 value from position 1

low = 0; 1 <= high <2 🡪 returns 2 values, position 0 & 1

0 < low <=1; high = 2 🡪 returns 2 values, position 1 & 2

low = 0; high = 2 🡪 returns 4 values, positions 1, 2 & 3