Draft appendix for S-98 Annex C Data loading and display algorithms.

Notes.

1. Taken from v1.3.0 of the S-101 Product Specification
2. Agreed by S-101PT 13 that it will migrate into S-98 Annex C
3. Comments required – also input on new algorithm proposed at S-101PT13
4. For inclusion in S-98 Annex C 1.3.0 onwards.
5. Appears to be out of date in respect of MaximumDisplayScale / optimumDisplayScale ?
6. Should any parts of 4.6 and 4.7 be included somewhere, particularly the highlighted section in red in 4.6 (which was discussed at PT13
7. Existing numbering has been preserved.

Introduction is constructed from parts of the existing text and references to parts of S-98 Annex C where parts already exist.

Introduction~~.~~

Details of the dataset loading and data display algorithms are contained within this Appendix.

Algorithms for dataset loading, ~~and~~ unloading~~;~~, and rendering (display) within a navigation system are prescribed for S-101 (and possibly other products which provide multiple generalizations for a given geographic area) in order for the appropriate ~~ENC~~ data to be viewed at the mariner's selected viewing scale.

Th~~is will~~ese algorithms are intended to ~~simplify the process for navigation systems, giving~~provide clear and concise rules on how and when data is loaded and unloaded; and the order at which datasets are to be displayed.

~~The concept of navigation purpose is restricted for use in presenting ENCs in a visual catalogue and must not be used for determining which dataset should be displayed, see clause XX-XX.~~

~~Details of the dataset loading and data display algorithms are contained within this Appendix.~~

Note 1: These algorithms only address loading and display related to visualization within the system graphics window. The application may need to load other datasets to satisfy requirements related to alerts processing, such as MSC.530(106) A11.2.

Note 2: Light sectors [and other features which may span dataset boundaries]. It should be possible, on request, for the mariner to be capable of identifying the colour of the sectors affecting the ship, even if the lights involved are off the display, and, in general provision must be made to ensure that all relevant features are portrayed.

Commented [DG1]: This seems unnecessary; the algorithms don't reference nav purpose

Commented [DG2]:
- Alerts must be evaluated using the best-scale dataset without regard for the MSVS.
- The OEM must ensure these datasets are loaded (the requirement is already in S-98 Annex C)

Commented [jp3]: These should be looked at in more detail.

Commented [DG4]:
- leg lines and sector arcs can extend far beyond the dataset boundaries.
- features such as text may also extend beyond the dataset boundary.
- The current loading algorithm doesn't account for this; it only uses the dataset boundaries.

4.6 Overview – Display scale ranges

Scales and scale ranges are defined in Clause XXX-XXX.

~~A~~The scale range of a dataset ~~is used to indicate a~~is the range of display scales between which a producer intends ~~considers~~ the data ~~is intended~~ for use. This scale range is ~~described~~determined by the scale ranges of the component **Data Coverage** features. Each **Data Coverage** feature has a **minimum display scale** and **maximum display scale**; the dataset scale range is from the smallest to the largest of these component values.

~~(See clause 4.7 for how datasets are to be loaded and unloaded within a navigation system.) The smallest scale is defined by the **minimum display scale** and the largest scale by the **optimum display scale**. The **maximum display scale** indicates the scale that the Data Producer considers that the "grossly overscaled" warning should be triggered. These scales must be set at one of the scales specified in clause 3 (spatial resolutions).~~

~~There must not be overlapping scale ranges (that is, overlaps between values of **optimum display scale** and **minimum display scale**) between datasets covering the same geographical area. ECDIS behaviour in the event of an overlap are dealt with in clause XX-XX.~~

~~*When the mariner's selected viewing scale (MSVS) is smaller than the value indicated by **minimum display scale**, features within the **Data Coverage** feature are not displayed, except where the System Database does not contain a dataset covering the area at a smaller scale, in which case the dataset will be continuously displayed.*~~

~~When the MSVS is larger than the value indicated by **optimum display scale**, the overscale indication, in the form of an overscale factor covering the area that is overscale, must be shown. When at own ship's position a dataset with a larger **optimum display scale** than the MSVS is available, an indication is required and must be shown on the same screen as the chart display. This is descrbied in Clause XX-XX~~

~~When the MSVS is larger than the value indicated by **maximum display scale**, the overscale indication, in the form of an overscale factor and, additionally, a pattern covering the area that is overscale, must be shown to indicate that the data is "grossly overscaled".~~

~~Within ENC schemes it is preferable that the scale ranges for different datasets covering the same geographical area to be continuous (see clause 4.5.3). However, where the scale ranges are non-continuous, the ECDIS will display the larger scale dataset until the MSVS is equal to or at smaller scale than the **optimum display scale** of the next smaller scale dataset This is described in Cause XX-XX.~~

ANNEX D – Dataset Loading and Display (Rendering) Algorithms ~~(Dataset Selection) and Dataset Display Order (Dataset Rendering)~~

~~**Preconditions**~~

~~An inventory for each **Data Coverage** contains:~~

- ~~A geo polygon describing the **Data Coverage**: *polygon(dataCoverage)*;~~
- ~~A set of scale bands: *scaleBands(dataCoverage)*;~~
- ~~An associated dataset: *dataset(dataCoverage)*.~~

~~A projection *projection* that can:~~

- ~~Convert geographic polygons *geoPolygon* to device polygons: *projection(geoPolygon)*;~~
- ~~Convert device polygons *polygon* to geographic polygons: ~*projection(polygon)*.~~

D-1 Dataset Loading Algorithm

D-1.1 Preconditions

An inventory for each **Data Coverage** contains:

Commented [DG5]: Datasets are intended for use between their minimum and maximum display scales (hence the attribute names). The optimum display scale just indicates an optimal viewing scale, it does not impact the dataset loading.

Commented [DG6]: Overscale warning is not part of loading / display

Commented [jp7]: All in definitions.

Commented [jp8]: Not for S98. But what to do in the event of an overlap should be.

Commented [DG9]: Encoding requirements are not part of loading / display

Commented [DG10]:
- This paragraph is incorrect.
- If the encoder wants the dataset to remain displayed they should encode MDS appropriately.
- The dataset will not be displayed once the user zooms out beyond the MDS.

Commented [DG11]: Not part of loading / display. Having requirements restated in different places leads to inconsistencies.

Commented [jp12]: All in S98 already.

Commented [jp13]: Intro?

Commented [DG14]:
- This is encoding guidance which is more appropriate for the DCEG.
- The guidance is incorrect; the smaller scale dataset is shown and the optimum display scale is not a factor in the selection.

Commented [DG15]:
- "Order" implies that the datasets are ordered. This is incorrect, the drawing instructions generated by the dataset portrayal are ordered.
- It's simplistic to indicate that the datasets themselves are "stacked" one atop another.

- A geo polygon describing the **Data Coverage**: *polygon(dataCoverage)*;
- A set of scale bands: *scaleBands(dataCoverage)*;
- An associated dataset: *dataset(dataCoverage)*.

A projection *projection* that can:

- Convert geographic polygons *geoPolygon* to device polygons: *projection(geoPolygon)*;
- Convert device polygons *polygon* to geographic polygons: *~projection(polygon)*.

~~D-10~~D-1.2 Scale Bands

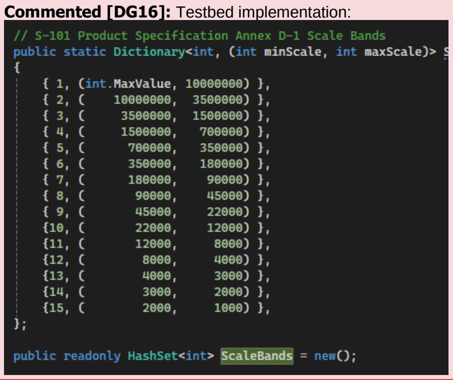
Attributes which represent scales store the denominator of the scale; an attribute value of “22,000” represents a scale of “1:22,000”. Larger values represent smaller scales, while smaller values represent larger scales.

A scale range is the set of scales between a minimum and maximum scale. The attributes **minimum display scale** and **maximum display scale** describe the scale range of a **Data Coverage** feature. The scale range of a dataset is a set of ranges, each range taken from a component **Data Coverage**.

Scale bands describe specific scale ranges, as shown in table XX. ~~A lists of scale bands will be~~is ~~used for by the algorithm.~~ Each scale band is defined by its minimum and maximum scale denominators and ~~will be~~is accessed ~~by~~ via an index. ~~Note that the table below contains the denominators of the scale; for example~~example,~~. 22,000 is the denominator value associated with the scale 1:22,000.~~ Note that ~~W~~whenever scales are compared in these algorithms the numerical comparison is based on scales, not on scale denominators.

index ***(scale band)***	***minimumScale***	***maximumScale*** ~~*(maximum)*~~	**Remarks**
1	NULL (∞)	10,000,000	For all values larger than 10,000,000
2	10,000,000	~~10,000,000 < *maximum* ≤~~ 3,500,000	
3	3,500,000	~~3,500,000 < *maximum* ≤~~ 1,500,000	
4	1,500,000	~~1,500,000 < *maximum* ≤~~ 700,000	
5	700,000	~~700,000 < *maximum* ≤~~ 350,000	
6	350,000	~~350,000 < *maximum* ≤~~ 180,000	
7	180,000	~~180,000 < *maximum* ≤~~ 90,000	
8	90,000	~~90,000 < *maximum* ≤~~ 45,000	
9	45,000	~~45,000 < *maximum* ≤~~ 22,000	
10	22,000	~~22,000 < *maximum* ≤~~ 12,000	
11	12,000	~~12,000 < *maximum* ≤~~ 8,000	
12	8,000	~~8,000 < *maximum* ≤~~ 4,000	
13	4,000	~~4,000 < *maximum* ≤~~ 3,000	
14	3,000	~~3,000 < *maximum* ≤~~ 2,000	
15	2,000	~~2,000 < *maximum* ≤~~ 1,000	

Commented [DG16]: Testbed implementation:



```
// S-101 Product Specification Annex D-1 Scale Bands
public static Dictionary<int, (int minScale, int maxScale)> S
{
    { 1, (int.MaxValue, 10000000) },
    { 2, (    10000000,  3500000) },
    { 3, (     3500000,  1500000) },
    { 4, (     1500000,   700000) },
    { 5, (      700000,   350000) },
    { 6, (      350000,   180000) },
    { 7, (      180000,    90000) },
    { 8, (       90000,    45000) },
    { 9, (       45000,    22000) },
    {10, (       22000,    12000) },
    {11, (       12000,     8000) },
    {12, (        8000,     4000) },
    {13, (        4000,     3000) },
    {14, (        3000,     2000) },
    {15, (        2000,     1000) },
};

public readonly HashSet<int> ScaleBands = new();
```

Commented [D17]: Note that because the mins are all shared there will never be “gaps” in the dataset scale range. The range is from the shared min to the largest (smallest value) max.

Commented [D18]: Needs table label

The following algorithm associates ~~a scale denominator with~~ an index (scale band) with a (display) scale:

```
Algorithm GetScaleBand(scale)
Input: A scale
Output The index of the scale band
    1. If scale < maximumScale[1]
        a. Return 1
    2. For index = 2 to 15
        a. If $minimumScale[index] <= scale\ AND\ scale\ < maximumScale[index]$
            i. Return index
    3. Return 15
```

The following algorithm associates a ~~The~~ set of scale bands with~~for~~ a **Data Coverage** feature~~with its~~

```
Algorithm scaleBands(dataCoverage)
Input: A Data Coverage
Output: A set of associated scale band indices S
    1. minimumDisplayScale – The minimum display scale of the coverage (if not defined it is
       assumed that the scale is 1:∞ -> 0)
       maximumDisplayScale – The maximum display scale of the coverage
    2. Create an empty set S
    3. If $minimumDisplayScale\ < maximumScale[1]$
        a. $S\ =\ S\ \cup 1$
    4. For index = 2 to 15
        a. If $max(minimumDisplayScale, minimumScale[index])\ <$
           $max(maximumDisplayScale, maximumScale[index])$
            i. $S\ =\ S\ \cup index$
    5. Return S
```

~~*minimumDisplayScale* and *maximumDisplayScale* is defined as~~:

~~D-11~~D-1.3 Dataset ~~Coverage~~ Selection Process

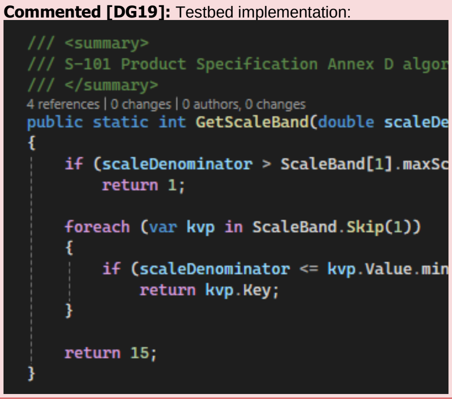
The ~~next~~ following algorithm ~~shows the selection process of the~~selects **Data Coverage** features. The output should be used to load each dataset which is associated with any of the selected features (*S*).~~s.~~

~~The idea is to~~The algorithm ~~find~~ evaluates ~~all~~ an inventory of **Data Coverage** features ~~for the scale band that contains the scale parameter~~ and selects those which overlap both the viewport and the indicated scale. If selected, ~~T~~the coverage footprint is subtracted from the viewport ~~should then be modified in a way that it only defines the part that is yet to be covered~~. This process is repeated until the viewport is empty or the entire inventory has been evaluated.

~~If this part is not empty the algorithm will proceed with the next smaller scale band until the remaining viewport is empty or there is no smaller scale band to investigate.~~

```
Algorithm SelectDataCoverages(inventory, scale, viewport, projection)
Input: An inventory of Data Coverage features inventory
        A scale for which the Data Coverage features will be selected (usually the display scale)
        A device-polygon viewport describing the device area that should be covered with data
        A projection projection
Output: A set of Data Coverage features S
    1. $S = \emptyset$
    2. $ScaleBand = GetScaleBand(scale)$
```

Commented [DG19]: Testbed implementation:

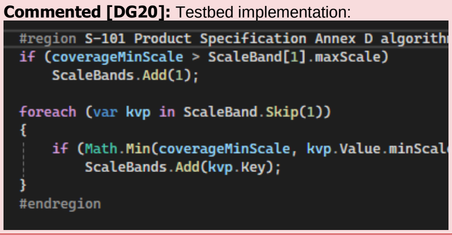


```
/// <summary>
/// S-101 Product Specification Annex D algor
/// </summary>
4 references | 0 changes | 0 authors, 0 changes
public static int GetScaleBand(double scaleDe
{
    if (scaleDenominator > ScaleBand[1].maxSc
        return 1;

    foreach (var kvp in ScaleBand.Skip(1))
    {
        if (scaleDenominator <= kvp.Value.min
            return kvp.Key;
    }

    return 15;
}
```

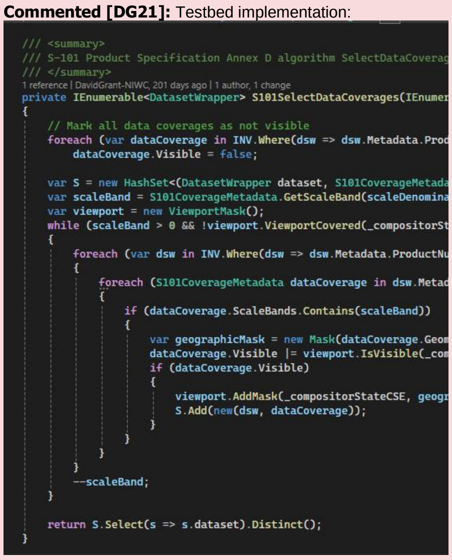
Commented [DG20]: Testbed implementation:



```
#region S-101 Product Specification Annex D algorith
if (coverageMinScale > ScaleBand[1].maxScale)
    ScaleBands.Add(1);

foreach (var kvp in ScaleBand.Skip(1))
{
    if (Math.Min(coverageMinScale, kvp.Value.minScal
        ScaleBands.Add(kvp.Key);
}
#endregion
```

Commented [DG21]: Testbed implementation:



```
/// <summary>
/// S-101 Product Specification Annex D algorithm SelectDataCoverag
/// </summary>
1 reference | DavidGrant-NIWC, 201 days ago | 1 author, 1 change
private IEnumerable<DatasetWrapper> S101SelectDataCoverages(IEnumer
{
    // Mark all data coverages as not visible
    foreach (var dataCoverage in INV.Where(dsw => dsw.Metadata.Prod
        dataCoverage.Visible = false;

    var S = new HashSet<(DatasetWrapper dataset, S101CoverageMetada
    var scaleBand = S101CoverageMetadata.GetScaleBand(scaleDenomina
    var viewport = new ViewportMask();
    while (scaleBand > 0 && !viewport.ViewportCovered(_compositorSt
    {
        foreach (var dsw in INV.Where(dsw => dsw.Metadata.ProductNu
        {
            foreach (S101CoverageMetadata dataCoverage in dsw.Metad
            {
                if (dataCoverage.ScaleBands.Contains(scaleBand))
                {
                    var geographicMask = new Mask(dataCoverage.Geom
                    dataCoverage.Visible |= viewport.IsVisible(_com
                    if (dataCoverage.Visible)
                    {
                        viewport.AddMask(_compositorStateCSE, geogr
                        S.Add(new(dsw, dataCoverage));
                    }
                }
            }
        }
        --scaleBand;
    }

    return S.Select(s => s.dataset).Distinct();
}
```

```
3. **While** $viewport \neq \emptyset$ **do**
    a. **For** all *dataCoverage* in *inventory*
        i. **If** $ScaleBand \in scaleBands(dataCoverage)$ AND $(projection(polygon(dataCoverage)) \cap viewport) \neq \emptyset$
            1. $S = S \cup dataCoverage$
            2. $viewport = viewport \setminus projection(polygon(dataCoverage))$
    b. $ScaleBand = ScaleBand - 1$
    c. **If** $ScaleBand = 0$
        i. **Return** *S*
4. **Return** *S*
```

Comments:

Row	Description
1.	Create an empty set of inventory **Data Coverage** features
2.	Get the scale band to which *scale* belongs and assign it to the variable *ScaleBand*
3.	~~As long as~~While the *viewport* area is not empty
3.a	Loop over all **Data Coverage** features in the inventory
3.a.i	If *ScaleBand* is an element of the scale bands of the **Data Coverage and** the projected coverage polygon of the **Data Coverage** overlaps the *viewport*
3.a.i.1.	Add the **Data Coverage** to *S*
3.a.i.2.	Remove the **Data Coverage** polygon from the *viewport*, The *viewport* will now only define the uncovered part of the original *viewport*
3.b.	Decrement *ScaleBand*
3.c.	If *ScaleBand* equals to zero (no scale band left to investigate)
3.c.i.	Return the collected result
4.	Return the collected result

~~Note that the algorithm above selects **Data Coverage** features. The system will then load the associated datasets. In the case where multiple selected **Data Coverage** features are associated with the same dataset, this dataset will be loaded only once.~~

~~D-12~~D-2 Data Display Algorithms

~~D-12.1~~D-2.1 Basic Data ~~display~~ Display ~~algorithm~~Algorithm ~~based on drawing index~~

~~D-12.1.1~~D-2.1.1 General

After the data-coverages are selected and the associated datasets are loaded the chart display will be generated by:

1. Create a set of drawing instructions for each dataset. This step is called portrayal and defined by the rules in the Portrayal Catalogue.
2. Render the drawing instructions as described below.

Notes:

- Datasets ~~can only~~must be portrayed entirely, there ~~is~~ must be no mechanism to only portray ~~single~~ selected data coverages.
- The algorithm assumes that the rendering is made by using a kind of the 'Painters algorithm'. This means an opaque fill will completely obscure what has been rendered at this position before. This does not mean that any implementation must follow this approach; other techniques like Z-Buffer

technique may be used. The algorithm will not give implementation details, any implementor has the freedom to reach the desired result in the most effective way.

~~D-12.1.2~~D-2.1.2 The Rendering Algorithm

The first step is to group the datasets into subsets which we will denote ‘Layers’. The criteria for the separation will be the ~~value for the attributes~~ **drawing index** and **minimum display scale** of the dataset (as derived from ~~for~~ the **Data Coverage** feature(s)~~ for the dataset~~). Note that all ~~**d**~~**Data** ~~**coverages**~~ **Coverage** features within a dataset must ~~have the same~~share common values for **minimum display scale** and **drawing index**;~~, and~~ data~~-~~sets with the same **minimum display scale** or the same **drawing index** are not ~~allowed~~ intended to overlap.

NOTE: in dual-fuel mode, the navigation purpose of S-57 datasets serves as a proxy for **drawing index**. S-57 datasets ~~should~~must be included as input to this algorithm when both products are enabled for simultaneous display. ~~To be precise, the union of all data coverages of one dataset must not overlap the union of the data coverages of another dataset with the same minimum display scale or drawing index.~~

A) Datasets which share a common (non-null) **drawing index** are grouped together in single layers.
   a. The **minimum display scale** of these layers is the smallest **minimum display scale** (the largest scale denominator) of the component datasets.

B) From the remaining datasets, those which share a common **minimum display scale** are grouped together in single layers.

C) Layers from A and B which share a common **minimum display scale** are grouped together in single layers.

The ‘Layers’ are then sorted by their **minimum display scale** and sequentially rendered ~~starting~~ from ~~with~~ the smallest **minimum display scale** to the largest.

Algorithm: *RenderChartImage(dataSets, viewport)*

Input: A set of datasets *dataSets* (previously selected using the dataset loading algorithm)

A device-polygon *viewport* describing the device area that should be covered with data

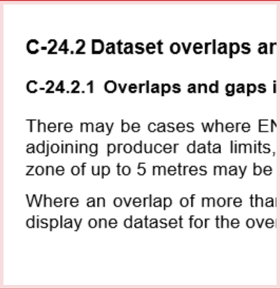
~~A drawing device~~

1. Fill *viewport* per *C-15.3.1 ENC No data areas*.
2. ~~Split~~Group ~~the set~~ *dataSets* by **drawing index** in~~in~~to ~~sub-sets denoted~~ *layer$_0$*, *layer$_1$*, …, *layer$_n$* (excluding those where **drawing index** is null or unknown)~~such that the **drawing index** of each dataset in one *layer$_x$* is not null but is otherwise the same~~.
   a. Assign a **minimum display scale** to each layer from the smallest **minimum display scale** (the largest scale denominator) of the component datasets.
3. ~~Split~~ Group the remaining dataSets (those where **drawing index** is null or unknown) ~~into sub-sets denoted~~by **minimum display scale** into *layer$_{n+1}$*, *layer$_{n+2}$*, … such that the **minimum display scale** of each dataset in ~~one~~ a given *layer$_x$* is the same.
4. Combine layers which share a common **minimum display scale**.
5. Sort the layers ~~*layer$_1$* .. *layer$_n$*~~ by ~~**its associated**~~ **minimum display scale**.
~~6. Clear the drawing device (e.g. by filling the drawing device with the NODTA colour or pattern.~~
~~7.~~ Iterate over ~~all~~ each *layer$_x$* ~~starting~~ from ~~with~~ the smallest **minimum display scale** to the largest.
6.
   ~~8.~~a. Render the layer to the *viewport* with the algorithm *RenderLayer*.~~Render the layer with the algorithm *RenderLayer*~~

NOTE 1: For the sake of ~~simplicity~~simplicity, the concept of display planes (that is, under and over radar) is not considered here. Without loss of generality the algorithm can be used multiple times to create the images for each display plane. One way of achieving it is to split the output of the portrayal into subsets; one for each display plane and run the algorithm for each subset. However, the painters algorithm cannot

Commented [DG22]: Per Note 1 below, the painters algorithm cannot be used when RADAR is enabled because the above RADAR display plane does not contain “skin of the earth” features / opaque fill. Features from underlying datasets will therefore remain visible.

Commented [DG23]:


C-24.2 Dataset overlaps ar

C-24.2.1 Overlaps and gaps i

There may be cases where EN
adjoining producer data limits,
zone of up to 5 metres may be

Where an overlap of more thar
display one dataset for the over

Commented [DG24]: The precise description is in the DCEG / PS be used to render data in the over radar display plane since there will not be Skin of the Earth objects present to obscure underlying layers.

NOTE 2: The algorithm as described here does not distinguish between official and non-official data. It could be achieved by taking this into account during the grouping of the input datasets.

~~D-12.1.3~~D-2.1.3 The ~~Algorithm~~ RenderLayer Algorithm

This algorithm describes how the datasets of one layer~~; that is, those that have~~ (sharing a single ~~the same~~ **minimum display scale**), ~~s and/or **drawing ind**ices~~ are to be rendered.

Algorithm: *RenderLayer**(dataSets, viewport)***

Input: A set of datasets *dataSets* that ~~have the same~~share a **minimum display scale** ~~and/or drawing indices~~

A device-polygon *viewport* describing the device area that should be covered with data~~A drawing device~~

1. **For** each ~~display~~ drawing priority *d**rawing**~~isplay~~Priority* starting with the smallest
    - a. From each dataset, ~~C~~collect ~~the~~ active[*] ~~–~~ drawing instructions ~~–~~ which are assigned to *drawingPriority*~~from each dataset's display instructions that are assigned to *displayPriority*~~.
    - ~~a.~~b. From the collection, render the instructions in the following order[**]:
        - ~~b.~~i. *~~Note: while~~ null instructions (NullInstruction) should not be rendered, but they should show in the pick report in this position (below the ~~the~~ other instructions which share the same drawing priority) ~~from that collection~~.*
        - ~~c.~~ii. all ~~Render the~~ area instructions ~~from that collection~~, followed by
        - ~~d.~~iii. all ~~Render the~~ line ~~instructions from that collection~~instructions, followed by
        - ~~e.~~iv. all ~~Render the~~ point instructions, followed by ~~from that collection~~
        - v. all ~~Render the~~ text instructions ~~from that collection~~

* The *viewingGroup(s), scaleMinimum, scaleMaximum, date dependency, line suppression, and any other* properties of the drawing instruction which may affect the instructions visibility must be taken into account. (See S-100 Part 9).

~~f.~~ ** To enhance the readability of text, an implementation may consider the guidance in S-100 Part 9 regarding text rendering to adjust this algorithm as needed.

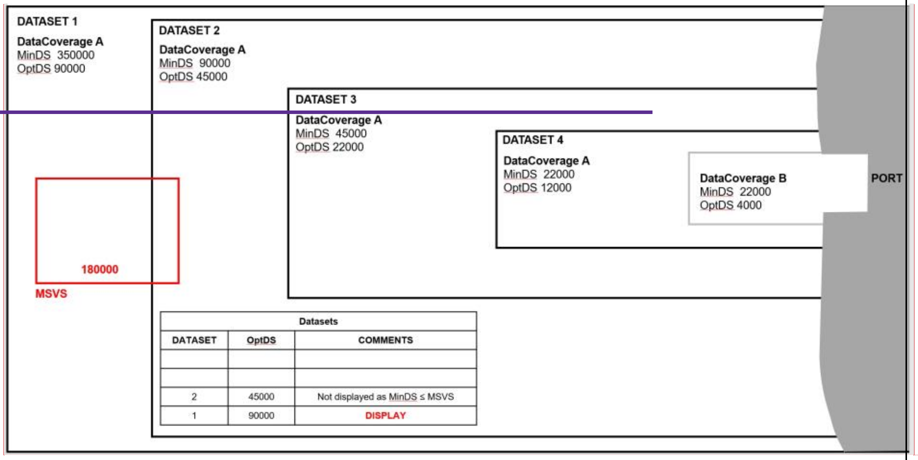
~~NOTES:~~

~~1a: Rendering must take the *viewingGroup(s)*, *scaleMinimum*, *scaleMaximum, date dependency, and any other* properties of the display instruction which may affect the instructions visibility into account. (See S-100 Part 9)~~

~~1f: When rendering text, an implementation may take into account the guidance in S-100 Part 9 regarding text rendering to adjust this algorithm as needed to enhance the readability of text.~~

~~4.7 Dataset loading and display order~~

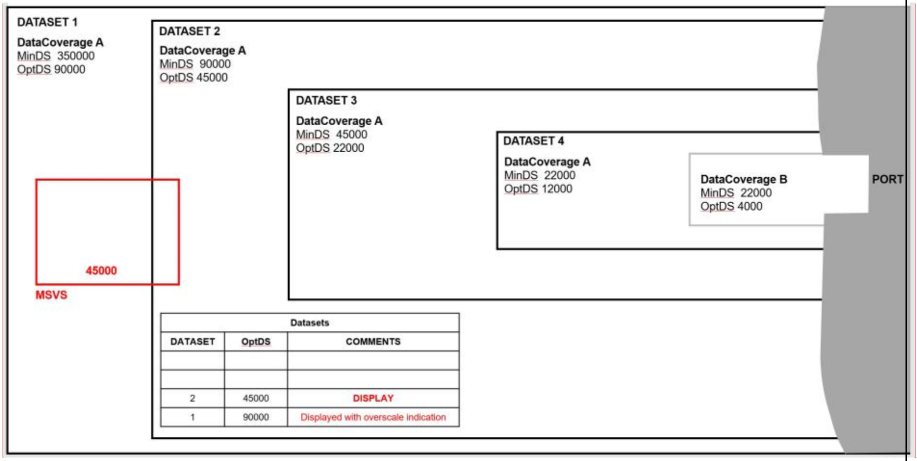
~~Figures 4-7 to 4-9 below are intended to assist in understanding how the datasets should be displayed in the system graphics window:~~



Datasets		
DATASET	OptDS	COMMENTS
2	45000	Not displayed as MinDS ≤ MSVS
1	90000	DISPLAY

~~Figure 4-7 – Dataset loading – scenario 1~~

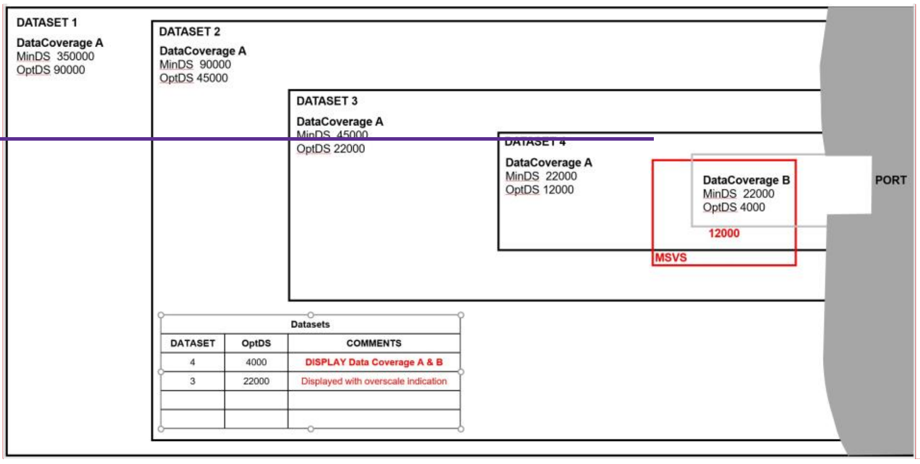
Commented [DG25]: Update to show maximum display scale rather than OptDS
- Also, update to show actual attribute names in all cases
- Also applies to remaining figures.



Datasets		
DATASET	OptDS	COMMENTS
2	45000	DISPLAY
1	90000	Displayed with overscale indication

~~Figure 4-8 – Dataset loading – scenario 2~~

Commented [DG26]: Remove overscale indication / pattern from data loading / display section



DATASET	OptDS	COMMENTS
4	4000	DISPLAY Data Coverage A & B
3	22000	Displayed with overscale indication

~~Figure 4-9 – Dataset loading – scenario 3~~

Commented [DG27]: DS4 DCA scale factor = 1x
DS4 DCB scale factor = 0.33x
DS3 DCA scale factor = 1.83x

D-2.2 ~~Alternate~~Advanced Data ~~d~~Display ~~a~~Algorithm

D-2.2.1 General

This algorithm is intended to resolve the following issues present in the basic algorithm described in~~ D-3.1~~ D-2.1:

- The basic algorithm cannot guarantee that smaller scale data does not obscure larger scale data.
- The basic algorithm ~~C~~cannot be used when rendering to the above-RADAR display plane.~~.~~
- In certain cases, the basic algorithm does not properly render drawing instructions which extend beyond the geographic boundary of the originating dataset.
  - For instance, portrayal of leg lines and sector arcs generated from Light features (when an adjacent dataset is rendered after the originating dataset).
- The basic algorithm may not work with future portrayal catalogs; it ~~relys~~relies on all “skin of the earth” features to hav~~ing~~e opaque fills.

The~~is~~ advanced algorithm addresses all these issues. It prioritizes rendering by drawing priority rather than by dataset;~~ by modifying the D-3.1 algorithm. It~~ ~~byres ring~~ it ensures drawing instructions originating from a visible point within a dataset remain visible as the instruction is rendered across adjacent datasets (even if those adjacent datasets obscure portions of the originating dataset); ~~I~~it does not rely on “skin of the earth” objects to obscure underlying information and can therefore be used to render to any display plane.

Commented [DG28]: Might want to add a paragraph describing why we’re providing two algorithms...

D-2.2.2 ~~Alternate~~Advanced Rendering Algorithm

The first step is to use information from the loading algorithm to assign a *mask* to each of the **Data Coverage** features within each dataset. Each *mask* represents the footprint of obscuring **Data Coverages;** it indicates areas of a dataset which should not be visible. Each dataset is then assigned a *mask* from the component **Data Coverage** features. The dataset *mask* is the union of the component **Data Coverage** masks.

```
Algorithm: ~~RenderChartImage~~AssignMasks(dataSets, viewport, projection)

Input:  A set of datasets dataSets (previously selected using the dataset loading algorithm)
        A device-polygon viewport describing the device area that should be covered with data
        A projection projection

  1. Collect all Data Coverage features from dataSets into dataCoverages.
  2. Sort dataCoverages by minimum display scale into sortedCoverages (from smallest to
     largest)
  3. For each dataCoverage in sortedCoverages
        a. Set mask of dataCoverage to Ø
        b. For each obscuringCoverage in sortedCoverages
              i. If obscuringCoverage <> dataCoverage AND scale(dataCoverage) <
                 scale(obscuringCoverage) AND projection(polygon(dataCoverage)) ∩
                 projection(polygon(obscuringCoverage)) ≠ Ø
                    ~~1.~~ dataCoverage.mask = dataCoverage.mask ∪
                       projection(polygon(obscuringCoverage))
              ~~1.Fill viewport per C-15.3.1 ENC No data areas.~~
     ~~1. Group dataSets by drawing index into layer$_0$, layer$_1$, …, layer$_n$ (excluding those where~~
     ~~drawing index is null or unknown).~~
     ~~a. Assign a minimum display scale to each layer from the smallest minimum display~~
     ~~scale (the largest scale denominator) of the component datasets.~~
     ~~1. Group the remaining dataSets (those where drawing index is null or unknown) by~~
     ~~minimum display scale into layer$_{n+1}$, layer$_{n+2}$, … such that the minimum display scale of~~
     ~~each dataset in a given layer$_x$ is the same.~~
     ~~1. Combine layers which share a common minimum display scale.~~
     ~~1. Sort the layers by minimum display scale.~~
     ~~1. Iterate over each layer$_x$ from the smallest minimum display scale to the largest.~~
                    1. ~~Render the layer to the viewport with the algorithm RenderLayer.~~
  4. For each dataset in dataSets
        a. Set mask of dataset to Ø
        b. For each dataCoverage in dataset
              ~~b.~~i. dataset.mask = dataset.mask ∪ dataCoverage.mask
```

The next step is rendering. The screen is filled with the no data pattern, then the active drawing instructions are collected from all loaded datasets and sorted by drawing priority. Each drawing instruction is then rendered using the *RenderInstruction* algorithm.

Algorithm: ~~*RenderChartImage*~~*RenderChartImage(dataSets, viewport)*

Input: A set of datasets *dataSets* (previously selected using the dataset loading algorithm)
A device-polygon *viewport* describing the device area that should be covered with data

1. Fill *viewport* per *C-15.3.1 ENC No data areas*.
2. Collect all active[*] drawing instructions from all *dataSets* into *drawingInstructions.*
3. Sort *drawingInstructions* by **drawing priority** into *sortedInstructions* (from smallest to largest)[**]
    a. Instructions which share a **drawing priority** must be ordered as follows:
        i. all null instructions, followed by
        ii. all area instructions, followed by
        iii. all line instructions, followed by
        iv. all point instructions, followed by
        v. all text instructions
4. For each *drawingInstruction* in *sortedInstructions*

~~1. *Fill viewport per C-15.3.1 ENC No data areas.*~~
~~1. *Group dataSets by **drawing index** into layer$_0$, layer$_1$, …, layer$_n$ (excluding those where **drawing index** is null or unknown).*~~
~~d. *Assign a **minimum display scale** to each layer from the smallest **minimum display scale** (the largest scale denominator) of the component datasets.*~~
~~1. *Group the remaining dataSets (those where **drawing index** is null or unknown) by **minimum display scale** into layer$_{n+1}$, layer$_{n+2}$, … such that the **minimum display scale** of each dataset in a given layer$_x$ is the same.*~~
~~1. *Combine layers which share a common **minimum display scale**.*~~
~~1. *Sort the layers by **minimum display scale**.*~~
~~1. *Iterate over each layer$_x$ from the smallest **minimum display scale** to the largest.*~~
a. ~~*Render the layer to the viewport with the algorithm RenderLayer.*~~*RenderInstruction(drawingInstruction)*

[*] The *viewingGroup(s)*, *scaleMinimum*, *scaleMaximum, date dependency, line suppression, and any other* properties of the drawing instruction which may affect the instructions visibility must be taken into account. (See S-100 Part 9).

~~a.~~ [**] To enhance the readability of text, an implementation may consider the guidance in S-100 Part 9 regarding text rendering to adjust this algorithm as needed.

D-2.2.3 The RenderInstruction Algorithm

This algorithm describes how each drawing instruction is to be rendered. Instructions originating from non-point geometries use *mask* to clip or mask the rendered output; those originating from occluded point geometries are not rendered, and those originating from non-occluded point geometries are rendered without masking or clipping.

Algorithm: *Rende*rInstruction~~rLayer~~(~~dataSets~~*drawingInstruction*, *dataset, viewport)*

Input: A ~~set of datasets~~ *d~~ataSets~~rawingInstruction* ~~that share a **minimum display scale**~~generated by dataset portrayal

The *dataset* of the drawing instruction

A device-polygon *viewport* describing the device area that should be covered with data

1. *isPoint* = false
    - a. ~~**For**~~ **If** *drawingInstruction* is an augmented point
        - i. *isPoint* = true
        - ii. *point* **=** augmented geometry
    - b. **Else If** geometry of *drawingInstruction* **feature reference** is a point
        - i. *isPoint* = true
        - ii. *point* = feature reference geometry
    - ~~2. **each drawing priority *drawingPriority* starting with the smallest**~~
2. ~~**From each dataset, collect active**[*] **drawing instructions which are assigned to *drawingPriority*.**~~**If** not *isPoint*
    - a. Render the drawing instruction, using the *dataset mask* to either clip or mask the rendered output. Portions of the rendered output which intersect *mask* should not be visible.

~~1.~~ **Else If** *dataset.mask* ∩ *point* <> Ø

3.
    - a. Do not render the instruction
4. **Else**
    - ~~a.~~ Render the instruction (without masking or clipping) ~~From the collection, render the instructions in the following order[**]:~~
        - ~~i. *null instructions (NullInstruction) should not be rendered, but they should show in the pick report in this position (below the other instructions which share the same drawing priority).*~~
        - ~~ii. all area instructions, followed by~~
        - ~~iii. all line instructions, followed by~~
        - ~~iv. all point instructions, followed by~~
        - ~~v. all text instructions~~

~~* Rendering must take the *viewingGroup(s)*, *scaleMinimum*, *scaleMaximum*, *date dependency, line suppression, and any other* properties of the display instruction which may affect the instructions visibility into account. (See S-100 Part 9).~~

a. ~~** To enhance the readability of text, an implementation may consider the guidance in S-100 Part 9 regarding text rendering to adjust this algorithm as needed.~~

S-98 Annex C – Annex D (Only)