**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 Appears to be out of date in respect of MaximumDisplayScale / optimumDisplayScale ?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, 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 data to be viewed at the mariner’s selected viewing scale.

These algorithms are intended to provide clear and concise rules on how and when data is loaded and unloaded; and the order at which datasets are to be displayed.

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.

## Overview – Display scale ranges

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

The scale range of a dataset is the range of display scales between which a producer intends the data for use. This scale range is 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.

# ANNEX D – Dataset Loading and Display (Rendering) Algorithms

1. **Dataset Loading Algorithm**
   1. **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)*.
  1. **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. Each scale band is defined by its minimum and maximum scale denominators and is accessed via an index. Note that whenever scales are compared in these algorithms the numerical comparison is based on scales, not on scale denominators.

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

The following algorithm associates 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]* 
   1. **Return** 1
2. **For** *index* = 2 to 15
   1. **If** 
      1. **Return** *index*
3. **Return** 15

The following algorithm associates a set of scale bands with a **Data Coverage** feature:

**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**
4. **For** index = 2 to 15
   1. If
5. **Return** S
   1. **Dataset Selection Process**

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

The algorithm evaluates an inventory of **Data Coverage** features and selects those which overlap both the viewport and the indicated scale. If selected, the coverage footprint is subtracted from the viewport. This process is repeated until the viewport is empty or the entire inventory has been evaluated.

|  |
| --- |
| **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 = ∅ 2. *ScaleBand* = 𝐺𝑒𝑡𝑆𝑐𝑎𝑙𝑒𝐵𝑎𝑛𝑑(𝑠𝑐𝑎𝑙𝑒) 3. **While** 𝑣𝑖𝑒𝑤𝑝𝑜𝑟𝑡 ≠ ∅ **do**    1. **For** all *dataCoverage* in *inventory*       1. **If** *ScaleBand* ∈ 𝑠𝑐𝑎𝑙𝑒𝐵𝑎𝑛𝑑𝑠(*dataCoverage*) AND (*𝑝𝑟𝑜jection*(*𝑝𝑜𝑙𝑦gon*(*dataCoverage*)) ∩ 𝑣𝑖𝑒𝑤𝑝𝑜𝑟𝑡) ≠ Ø          1. 𝑆 = 𝑆 ∪ *dataCoverage*          2. 𝑣𝑖𝑒𝑤𝑝𝑜𝑟𝑡 = 𝑣𝑖𝑒𝑤𝑝𝑜𝑟𝑡 \ *𝑝𝑟𝑜jection*(𝑝𝑜𝑙*ygon*(*dataCoverage*))    2. *ScaleBand* = *ScaleBand* – 1    3. **If** *ScaleBand* = 0       1. **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.** | 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 |

1. **Data Display Algorithms**
   1. **Basic Data Display Algorithm**
      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 must be portrayed entirely, there must be no mechanism to only portray 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.
  + 1. **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 **drawing index** and **minimum display scale** of the dataset (as derived from the **Data Coverage** feature(s)). Note that all **Data Coverage** features within a dataset must share common values for **minimum display scale** and **drawing index**; datasets with the same **minimum display scale** or the same **drawing index** are not 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 must be included as input to this algorithm when both products are enabled for simultaneous display.

1. Datasets which share a common (non-null) **drawing index** are grouped together in single layers.
   1. The **minimum display scale** of these layers is the smallest **minimum display scale** (the largest scale denominator) of the component datasets.
2. From the remaining datasets, those which share a common **minimum display scale** are grouped together in single layers.
3. 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 from 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

1. Fill *viewport* per *C-15.3.1 ENC No data areas*.
2. Group *dataSets* by **drawing index** into *layer0*, *layer1*, …, *layern* (excluding those where **drawing index** is null or unknown).
   1. Assign a **minimum display scale** to each layer from the smallest **minimum display scale** (the largest scale denominator) of the component datasets.
3. Group the remaining dataSets (those where **drawing index** is null or unknown) by **minimum display scale** into *layern+1*, *layern+2*, … such that the **minimum display scale** of each dataset in a given *layerx* is the same.
4. Combine layers which share a common **minimum display scale**.
5. Sort the layers by **minimum display scale**.
6. Iterate over each *layerx* from the smallest **minimum display scale** to the largest.
   1. Render the layer to the *viewport* with the algorithm *RenderLayer*.

NOTE 1: For the sake of 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 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.

* + 1. **The RenderLayer Algorithm**

This algorithm describes how the datasets of one layer (sharing a single **minimum display scale**)**,** are to be rendered.

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

**Input**: A set of datasets *dataSets* that share a **minimum display scale**

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

1. **For** each drawing priority *drawingPriority* starting with the smallest
   1. From each dataset, collect active\* drawing instructions which are assigned to *drawingPriority.*
   2. From the collection, render the instructions in the following order\*\*:
      1. *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).*
      2. all area instructions, followed by
      3. all line instructions, followed by
      4. all point instructions, followed by
      5. all text instructions

\* 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).

\*\* 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.

* 1. **Advanced Data Display Algorithm**
     1. **General**

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

* The basic algorithm cannot guarantee that smaller scale data does not obscure larger scale data.
* The basic algorithm 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 relies on all “skin of the earth” features to have opaque fills.

The advanced algorithm addresses all these issues. It prioritizes rendering by drawing priority rather than by dataset; 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); 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.

* + 1. **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**: *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*
   1. **Set** *mask of dataCoverage to* Ø
   2. **For** each *obscuringCoverage* in *sortedCoverages*
      1. **If** *obscuringCoverage <> dataCoverage* AND *scale(dataCoverage) < scale(obscuringCoverage)* AND *projection(polygon(dataCoverage)) ∩ projection(polygon(obscuringCoverage)) ≠ Ø*
         1. *dataCoverage.mask* = *dataCoverage.mask* ∪ *projection*(*polygon*(*obscuringCoverage*))
4. For each *dataset* in *dataSets*
   1. Set mask of dataset to Ø
   2. For each *dataCoverage* in *dataset*
      1. *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(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)\*\*
   1. Instructions which share a **drawing priority** must be ordered as follows:
      1. all null instructions, followed by
      2. all area instructions, followed by
      3. all line instructions, followed by
      4. all point instructions, followed by
      5. all text instructions
4. For each *drawingInstruction* in *sortedInstructions*
   1. *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).

\*\* 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.

* + 1. **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**: *RenderInstruction(drawingInstruction, dataset, viewport)*

**Input**: A *drawingInstruction* 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
   1. **If** *drawingInstruction* is an augmented point
      1. *isPoint* = true
      2. *point* **=** augmented geometry
   2. **Else If** geometry of *drawingInstruction* **feature reference** is a point
      1. *isPoint* = true
      2. *point =* feature reference geometry
2. **If** not *isPoint*
   1. 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.
3. **Else If** *dataset.mask* *∩ point* <> Ø
   1. Do not render the instruction
4. **Else**
   1. Render the instruction (without masking or clipping)