Inline Viewer Technical Guide

Version 1.0.0.92, 19 July 2018

1 Background

The Inline Viewer ("ixviewer") is a browser-hosted application for viewing an Inline XBRL document and all of its XBRL metadata. A critical usability feature of the application is that it does not require the end user's browser to validate or otherwise process the entire DTS, which, for some EDGAR submissions, may be slow. Instead, ixviewer relies on a plugin, called EDGAR Renderer, to preprocess all the information into a single compact data set called a MetaLinks file.

Detail about downloading, installing and running an "end to end" EDGAR Renderer that takes EDGAR Inline XBRL submissions, produces all the outputs needed for viewing, and starts up a small local http server for viewing the files, is available at these links:

EDGAR Renderer installation	http://arelle.org/documentation/edgar-renderer-installation/
EDGAR Renderer Source Code	http://arelle.org/download2/applications2/
EdgarRenderer	https://github.com/Arelle/EdgarRenderer/
EDGAR Renderer Technical Operation	http://arelle.org/documentation/edgar-renderer-installation/edgar-renderer-technical-operation/

2 Goal

The purpose of this document is to explain the overall process, the MetaLinks file, and the main components of the Inline viewer's implementation in JavaScript. The audience is assumed to be technical and familiar with XBRL, HTML and JavaScript.

Table of Contents

3 Processing Overview

The figure below conveys these key points:

1. All of the inputs to the process must be in a single folder. That may be a logical folder in a zip archive having no internal sub folders.
    a. A feature of the renderer is that if it is not given a specific file to process,and finds more than one Inline XBRL and/or Instance in the folder, it processes them all into a single combined rendering output.
2. The Edgar Renderer is a single application that includes a validation step ("EDGAR Filer Manual" or EFM Validation), which in turn relies on the Arelle XBRL processor.
    a. By default, a file that fails to validate *may* nevertheless be processed for rendering, but only for debugging. EDGAR itself is not configured to do so; EDGAR submissions that do not validate do not produce renderings, and vice versa.
3. There are several output files, depending on the type of rendering being performed, but they all are saved in a single output folder, whose default name is "out".

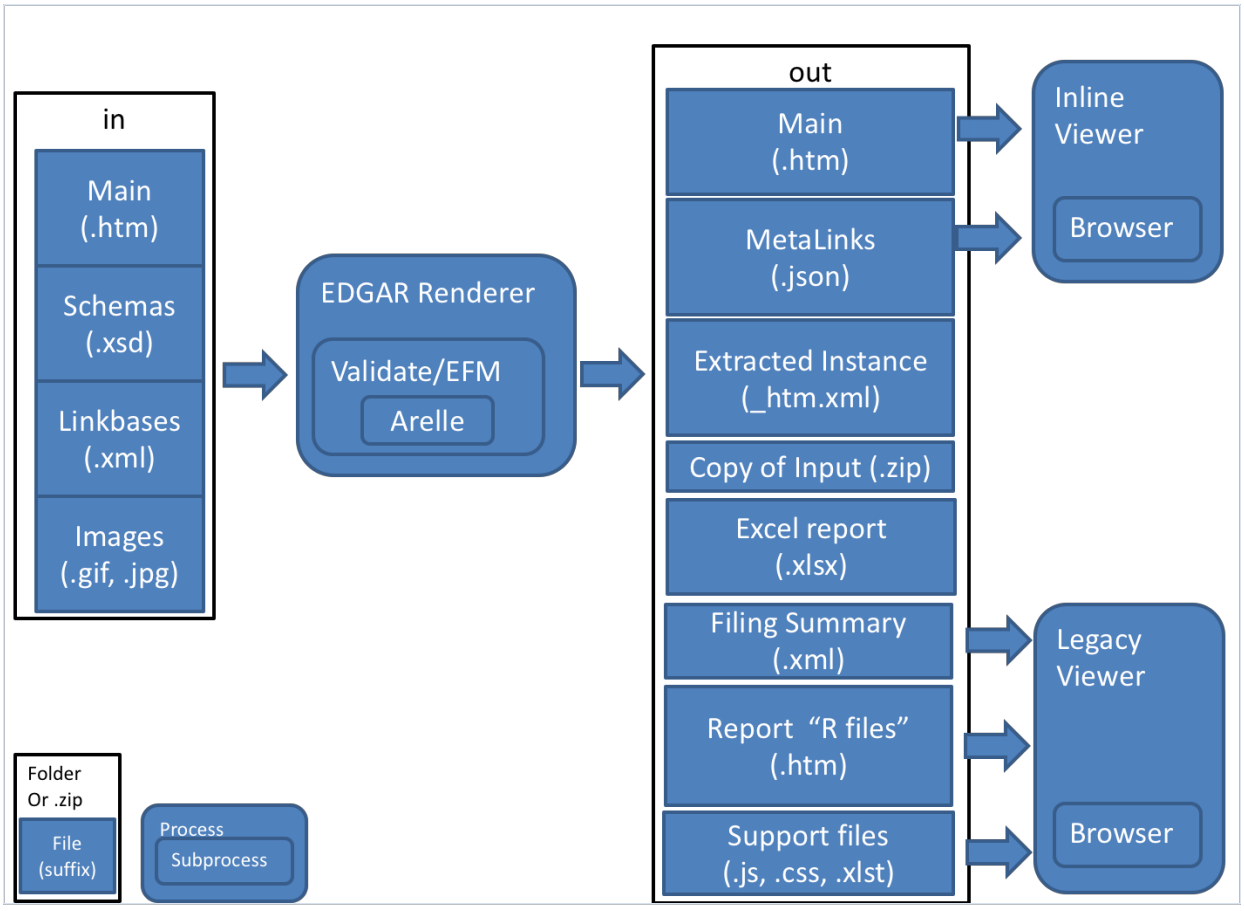

Figure 1. Processing overview showing the relationship of EDGAR Renderer to the Viewers

There are three groups of files in the output.

1. Outputs used by the Inline Viewer, consisting of:
    a. The main .htm Inline XBRL document and the .gif and .jpg images referenced in the file; all are simply copied from the input to the output.
    a. The MetaLinks.json file accompanies it, and contains all of the metadata assembled from the Inline XBRL file's Discoverable Taxonomy Set (DTS).
2. Outputs used by the Legacy Viewer, described in EFM 6.24 and 6.25 and consisting of:
    a. The filing summary, an xml file that contains the list of all individual html pages
    b. One or more "R files", each representing a single XBRL presentation group
    c. Any .png files produced from rendering risk/return summary data
    d. Several supporting files to support navigation, styling, and display of warning and error messages found in the filing summary.
4. Outputs used for data consumption
    a. The Inline XBRL file is processed to produce an "extracted instance", that is, an ordinary XBRL instance document containing all, and only, the XBRL content of the main file.
    b. A copy of the input files, packaged into a single zip archive.
    c. An Excel .xlsx file containing one worksheet for each of the Report files.

Risk/Return summary XBRL files do not produce an Excel output, and legacy XBRL instance documents do not produce MetaLinks. Command line options on the renderer can control whether or not each output is produced. Additional details regarding the conditions under which EDGAR produces each such file may be found in the Public Dissemination Service (PDS) technical guide:

https://www.sec.gov/info/edgar/specifications/pds_dissemination_spec.pdf

The remainder of this document covers only the Inline Viewer.

4 The MetaLinks file

MetaLinks is a JSON file, structured as nested levels of objects (whose members are unordered) and arrays (whose members are ordered). The atomic components of the JSON file are always strings.

MetaLinks contains a "compiled" version of all the information needed from the DTS of the Inline XBRL document, as follows:

- "version". The version of the MetaLinks file format, currently "2.0".
- "std_ref". An object whose member are all the standard references used in the instance. The object uses arbitrary keys "r0", "r1", "r2", etc. Standard references play an essential role in the Inline Viewer. They are not only visible on each associated XBRL fact, but are also searchable. Each reference entry is itself an object whose keys correspond to the names of reference linkbase roles allowed by EDGAR:
    - "Glossary"
    - "Name"
    - "Paragraph"
    - "Publisher"
    - "Section"
    - "SubTopic"
    - "Topic"
    - "URI"

- "instance". An object containing all the instances found, with the filename being the key and the value being a dictionary with a number of summary data points and other objects.
    - Summary strings:
        - "nsPrefix", "nsUri", the prefix and namespace used in the main custom schema of the instance
        - "contextCount", the number of contexts used in the instance
        - "entityCount", the number of distinct entity element values (for EDGAR, always 1)
        - "segmentCount", the number of distinct combinations of axis and member in contexts
        - "keyCustom", "keyStandard", the number of custom and standard item elements
        - "memberCustom", "memberStandard", the nmber of custom and standard member elements used.
        - "axisCustom", "axisStandard", the number of custom and standard axis elements used.
    - A "hidden" facts summary object, with keys as namespace URI's and the number of facts in each namespace the value. For example:

```
"hidden": {
 "http://fasb.org/us-gaap/2018-01-31": 2,
 "http://xbrl.sec.gov/dei/2018-01-31": 6,
 "total": 8
        },
```

    - A "dts" object, with the key being the document type, and an array of local and remote URLs, respectively. For example, the DTS of this instance contains one local and two remote label linkbases:

```
"dts": {
    "labelLink": {
                "local": [
                        "bst-20160930_lab.xml"
                    ],
                "remote": [
  "http://xbrl.fasb.org/us-gaap/2018/elts/us-gaap-doc-2018-01-31.xml",
  "http://xbrl.sec.gov/dei/2018/dei-doc-2018-01-31.xml"
                            ]
                    },
```

    - A "tag" object, where, for each element id as key, the value is an object with all the metadata relevant to the Inline Viewer as derived from processing the DTS:
        - "localname", the local-name() of the element QName;
        - "nsuri", the namespace-uri() of the element QName;
        - "auth_ref", an array of references,
        - "presentation", an array of roleType URI's where the fact is presented;
        - "calculation", an object with keys as roleType URI's where the element has a calculation parent;
        - "xbrltype", with values such as "domainItemType", "monetaryItemType", etc.
        - "crdr", with value "credit" or "debit".

      For example, here is a custom XBRL element as it appears appearing in a "tag" object:

```
"bst_PaymentsForShareBasedAwardsExercisedNetOfTaxesPaid": {
 "auth_ref": [],
 "calculation": {
  "http://www.brightstar.com/taxonomy/role/StatementOfCashFlowsIndirect": {
```

```
      "order": 1.13,
      "parentTag": "us-gaap_NetCashProvidedByUsedInFinancingActivities",
      "weight": -1.0
     }
    },
    "crdr": "credit",
    "lang": {
     "en-US": {
      "role": {
       "documentation": "Payments for share based awards, net of taxes paid.",
       "label": "Payments For Share Based Awards Exercised Net Of Taxes Paid",
       "negatedLabel": "Share-based awards exercised, net of taxes paid"
      }
     }
    },
    "localname": "PaymentsForShareBasedAwardsExercisedNetOfTaxesPaid",
    "nsuri": "http://www.brightstar.com/20160930",
    "presentation": [
     "http://www.brightstar.com/taxonomy/role/StatementOfCashFlowsIndirect"
    ],
    "xbrltype": "monetaryItemType"
   },
```


- Finally, a "report" object. The R-file base name is used as a key, "R1", "R3", "R9999", etc. Most of the data items are equivalent to the contents of the Filing Summary xml file:
  - "role", the role URI of the presentation group.
  - "longName", as documented in EFM 6.7.12, consisting of a sequence of digits, a group type, and a name indicating the content of the R-file.
  - "groupType", with value one of "document", "disclosure", "statement", or "schedule" as in EFM 6.7.12.
  - "subGroupType", with value one of "details", "tables", "policy", or "parentheticals" as described in EFM 6.24.20.
  - "isDefault", a Boolean that indicates whether when viewing the rendering, this is the first report to be opened in the browser.
  - Each "report" object also has two keys with information that is entirely specific to Inline Viewer navigation, "firstAnchor" and "uniqueAnchor". These are used to locate presentation groups in the Inline XBRL document.

5 Locating Presentation Groups in the Inline XBRL document

An Inline XBRL document is a well-formed XHTML document in which additional elements from the "ix" namespace have been interleaved. For the Inline Viewer to locate any ix:nonFraction or ix:nonNumeric element within the source document, it is usually sufficient to know the combination of element name, context id, and, if present, the unit id or language token. Then, having located the fact, it is sufficient to find the first preceding (or, in general, any block type HTML element) and scroll to that location. As of MetaLinks version 2.0, the MetaLinks file does not need, and does not contain, a separate JSON object for every fact in the instance.

This is not true of locating presentation groups. In principle, one might expect that a role type called "000 – Statement – Balance Sheet" could be located in the original HTML by (say) searching for the words "Balance Sheet". This is not robust, however. It does not work when, for example:

- A balance sheet is spread across two pages with a in the middle; when
- The contents of a single Note to the Financial Statements is tagged at multiple levels of detail as required by EFM 6.7.12 and 6.24.20; when

- ix:continuation elements connect widely separated elements, and so on.

Despite EFM 6.13's guidance directing filers to create presentation groups and presentation relationships reflecting the order in the Original HTML/ASCII document, the ideal mapping between an HTML file location and the content of an R-file for a presentation group is necessarily approximate.

Therefore, during the rendering process all of the facts in each presentation group are analyzed to determine (a) whether that is the only presentation group where the fact is "uniquely" rendered, and (b) what is the "first" fact of the presentation group to appear in the HTML file. Either can work as an "anchor fact" to find the immediately preceding . But since the rendering process cannot modify the original HTML file, to (say) add an attribute or identifier to mark such an anchor location, the MetaLinks file instead stores the location of the fact in the form of an ancestor list of HTML elements for the fact.

The Inline Viewer can navigate to a most-likely correct location in the original HTML file, based on either a unique fact *or* the first fact. The algorithm searches backward through HTML sibling elements, and up through the ancestor list, skipping over elements that do not match, until it finds an element that is displayed as a block (again, usually a).

The data stored in the MetaLinks file for each report to support this algorithm is:

- "firstAnchor", an object with the "ancestors" that is an array of element names; and
- "uniqueAnchor", also an object of "ancestors", but also containing a key for contextRef and a key "first" that is "true" if the unique anchor is the same as the first anchor.

For example, consider a presentation group in which there is an XBRL Table Text Block element (us-gaap:XyzTableTextBlock) located within a level 1 note (us-gaap:XyzTextBlock). The resulting report object contains the ancestor path along with keys and values to uniquely locate the fact:

```
"R9": {
     "longName": "42 - Disclosure – Xyz Disclosure (Tables)",
     "firstAnchor": {
      "ancestors": [
       "us-gaap:XyzTextBlock", "div", "body", "html"
      ],
     "uniqueAnchor": {
      "ancestors": [
       "us-gaap:XyzTextBlock ", "div", "body", "html"
      ],
      "contextRef": "NineMonthsEnded_2016-09-30",
      "decimals": null,
      "first": true,
      "lang": "en-US",
      "name": "us-gaap:XyzTableTextBlock",
      "reportCount": 1,
      "unique": true,
      "unitRef": null,
      "xsiNil": "false"
    }}
```

6 Inline Viewer Components

Inline viewer requires the following minimum browser versions:

- Internet Explorer 10 or greater
- Chrome 27 or greater
- Safari 7 or greater

- Mozilla 22 or greater

The inline viewer uses jquery 2.1.1 and the bootstrap 3.2.0 framework.

It has these main components:

1. A root file, ix.html, which contains the static HTML content, with a single <iframe> element that will contain the Inline document;
2. A Cross Browser Extension (CBE) object that is instantiated separately in each browser window;
3. An App object that defines the data structures and accessors for the Inline document and XBRL metadata related functions;
4. A Find module for all of the application's functions related to searching, filtering, and highlighting; and
5. A Samples folder that is always included in the distribution.

There are a number of smaller components to be described individually later.

6.1 Root (ix.html)

The root file, ix.html, first checks for a compatible browser version and shows an alert with instructions to the user to use a newer browser. It then loads all the JavaScript and CSS modules. When loading, there is a query parameter added to the URL that reflects the current version number, so as to ensure a cache miss when a new version of the Inline Viewer is deployed. For example, in version 1.0.0.93:

```
<link type="text/css" rel="stylesheet" href="css/rightNavigation.css?v93" />
<script type="text/javascript" src="js/app/app.js?v93"></script>
```

All of the static HTML content for the static menu bar, dialogs, help content, etc., appears in this file.

The viewer expects the ix.html URL to be opened with a query parameter, "doc", that contains the URL of the Inline file to be viewed. Here is a typical invocation URL:

http://localhost/ixviewer/ix.html?doc=samples/bst/out/bst-20160930.htm

The inline XBRL file is loaded in an <iframe>.

Note that for most browsers, simply opening ix.html from the filesystem will not work; it must be accessed via http or https. For example, copying the ixviewer folder to the C: drive and trying to use this url not work:

file:///c:/ixviewer/ix.html?doc=samples/bst/out/bst-20160930.htm

Depending on your browser, it might simply fail to load with no explanation, or it might display the message:

Can not find file (samples/bst/out/bst-20160930.htm).

Opening it in Chrome with the Develoepr Tools and examining the console log, you will see that it is a CORS violation:

```
jquery-2.1.1.min.js?v93:4 Failed to load file:///C:/ixviewer/samples/bst/out/bst-
20160930.htm: Cross origin requests are only supported for protocol schemes: http,
data, chrome, chrome-extension, https.
```

As noted in section 1 above, the Arelle downloadable GUI takes care of setting up an http server for the sole purpose of viewing Inline XBRL files.

The SEC web site does not permit cross origin requests, either. Therefore, a locally installed viewer will not be able to load a file directly from the SEC site. Rather than leave such a CORS error uncaught, the Inline viewer requires that the URL scheme, host name and protocol must be identical:

> The protocol, host name and port number of the 'doc' field (http://www.sec.gov/ixviewer/samples/bst/out/bst-20160930.htm), if provided, must be identical to that of the Inline XBRL viewer (http://localhost)

In general, therefore, the doc parameter will be a relative URL.

Errors are also shown if the doc parameter is absent, if a HEAD request to the document url returns an error, or if the MetaLinks.json file is not found at the same path as the html file.

Finally, note that there is a slightly different version of the root file, called ix_softlink.html, that is used on the www.sec.gov server, which redirects "/ix" to a copy of "/ixviewer/ix_softlink.html". Thus, on www.sec.gov the URL looks like this and the relative paths of the JavaScript and other files are adjusted accordingly:

https://www.sec.gov/ix?doc=ixviewer/samples/bst/out/bst-20160930.htm

6.2 CBE (js/cbe.js)

Each window of the inline viewer has an instance of the cbe. This object contains the data structures created during the loading of the Inline XBRL file. Among the objects cached during this process:

- The MetaLinks object as loaded from MetaLinks.json;
- The contents of the ix:header and ix:hidden elements, lists of contexts, context IDs, dimensions relationships; and
- An array for each sequence of ix:continuation elements.

6.3 App (js/app.js)

The App object contains all the behaviors associated with the various buttons, menus and dialogs. Its most important function, however, is to initiate asynchronous loading of the Inline XBRL document, having established that it exists via the HEAD request executed by ix.html.

A typical EDGAR inline XBRL document may be a very large HTML file of up to 200 MB, where only the last set of pages – the Financial Statements and Notes– contains any significant number of inline XBRL elements. The asynchronous load allows users to begin viewing the HTML content while the application processes the inline XBRL elements and constructs its caches and indexes to support navigation.

The App object is where the version of the application is stored.

App also contains various XBRL-specific functions such as the format transformation functions, which use the "format" and "scale" attributes of ix:nonFraction and ix:nonNumeric to transform strings such as "June 6, 2019" into the ISO 8601 string "2019-06-06" and "1,234" with scale of 3 into "1234000".

6.4 Find (js/find.js)

The find module contains all of the logic to interpret the user's choices related to highlighting and displaying facts in the fact pane:

- Using the Settings dialog to determine whether searches should include labels, definitions, dimensions, references, in addition to element names, and whether to match case explicitly;

- Using the Data filter to determine whether to show all elements, or only those that are Amounts, Text, those with Calculations, those with negative values, or elements appearing in the "ix:hidden" section, referred to in the application as "Additional Items".
- Using the Tags filter to determine whether to include custom, standard, or both types of elements; and
- Several other filters including periods, numeric measures, specific axes and members, scales, or elements with or without credit/debit balances.

Among the most complex functions in the find module are those dealing with the highlighting of text appearing on the screen. In essence, the highlighting function is manipulating the Document object loaded from the Inline XBRL file in the following ways:

- Where an appropriate element exists that contains all, and only, the text of the fact to be highlighted it manipulates its class and other attributes, resulting in the CSS properties of that class to be applied. For example, on a numeric element:
    - A fact that appears in a search result gets the tokens "sec-cbe-highlight-*inline*" and "sec-cbe-highlight-dashed" added to class attribute of its element.
    - If it is the 5th hit in a search list, it gets a new attribute "data-result-index" set to 5.
    - While the mouse is over it, it gets token sec-cbe-highlight-filter-content-selected added to the class.
    - When the user clicks on it to bring up the attributes dialog, the token sec-cbe-highlight-filter-selected is added to the class.
- Where the original Inline document did not have an appropriate element, during the load step, one is inserted into the Document object as a "wrapper", so that the necessary class attributes can be set.
- On a block text element:
    - A fact that appears in a search result gets the token "sec-cbe-highlight-*block*" instead, to indicate that the HTML layout method is "block" and not "inline".
    - A small image of a rectangle is inserted into the Document as the first child of the as a visual marker.
    - Each color allowed by the Settings dialog has its own separate image file; the default orange color, for example, is file "ff6600_img.png". Loading the tiny image and then setting its horizontal and vertical dimensions is faster and more reliable than (say) trying to insert a table cell with that same orange background. There is a separate file for each of the 216 web safe colors in the /images folder.

6.5 Samples

The samples folder is considered part of the distributed inline viewer. It illustrates the usual relationship between an inline file, its supporting XBRL schema and linkbases, and the outputs that are normally produced by the Renderer.

At this time there is a single example.

- Folder "bst" contains a BlueStar Airlines 3rd quarter 10-Q. fragment with only the statements and two notes. File bst/bst-201609.htm has custom elements, calculations, some facts in ix:hidden, dimensions, and a use of the ix:exclude element to skip over a page break. Sub folder "out"

contains the MetaLinks.json file, the extracted instance bst-20160930_htm.xml, and the zip file containing all inputs.

- The "Sections" menu item illustrates navigation to presentation groups.
- It does not illustrate continuations nor the use of image files in the input folder.

6.6 Other components

Polyfills (js/app/polyfills.js) defines functions that may be absent from a given browser's set of built in functions.

Errors (js/app/errors.js) contains custom error handlers and messages.

URI (js/app/uri.js) implements a URI object used for interpreting the doc parameter.

These files will be removed from a future distribution; they are no longer used:

- FileSaver.js
- jszip.min.js
- In folder /images:
  - Blue-vertical-rectangle.png
  - Copy.png
  - Copyold.png
  - File-open.png
  - Filter.png
  - Help.png
  - Info.png
  - Orange-vertical-rectangle.png
  - Save-as.png
  - Securitypagebackground.jpg
  - Settings.png