Data Galaxies: How to Use

Adapted for the DataGalaxies app for webAppOS.

Sergejs Kozlovičs

# Introduction

## Target Audience

This document contains guidelines for three groups of people:

* data galaxy end-users, who will use the Data Galaxy tool in the end-user mode;
* data galaxy designers, who will create data galaxies from available galaxy components (stars, stellar winds, planets, and filters corresponding to data sources, data transformations, data visualizations, and cross-filters);
* developers, who will create new galaxy components (stars, stellar winds, planets, and filters) that can be used by designers and end-users.

## The Structure of the Document

The document consists of four chapter. The first one contains general information that is useful for all groups of the users. Chapters 2-4 are addressed (respectively) to end-users, designers, and developers. While end-users may stop reading after Chapter 2, designers should continue reading until Chapter 3, while developers are expected to read until the end of the document.

This document does not include documentation webAppOS, a technology beyond DataGalaxies. For that documentation refer to <http://webappos.org>.

# General Information

## Opening/Creating a Data Galaxy Project

To open an existing project (either designer, or end-user project):

* Launch the DataGalaxies app (e.g., from webAppOS desktop), click “Browse”, and point to the desired .datagalaxy file.
* Optionally, you may upload an existing .datagalaxy file using the webAppOS FileBrowser app.

To create a new end-user project or a new designer project based on some predefined template (we discuss the process of deploying end-user templates in Section 3.7):

* Launch the DataGalaxies app (e.g., from webAppOS desktop). The available templates appears below the “Browse” button in sections “New from template” and “New from user template”. These sections contain global templates provided by the webAppOS administrator, as well as user templates stored in your home templates directory, respectively. If there are some data galaxies preconfigured and deployed to run in the end user mode, they will be appear among predefined designer-mode templates. Choose the desired template, and it will be used as a basis for your new project.

To create a new data galaxy project in the designer mode:

* Launch the DataGalaxies app (e.g., from webAppOS desktop) and choose the “Bootstrap new” option.

Based on the designer’s project, the designer will be able to deploy templates for end users.

## Galaxy Components Inside

Each data galaxy consists of stars (data sources), stellar winds (data transformations), planets (data visualizations), and cross-filters, which can affect certain transformations and visualizations. Inside the galaxy, data can be transformed multiple times until the desired result, which can be visualized, is reached. While the galaxy designer can configure and control the whole process, end users usually do not need to know how the data is transformed. Thus, when deploying the galaxy to end users, the designer usually hides certain components. Still, the users may be able to configure some parts of the galaxy. For instance, the user may be allowed (or even required) to provide the source data file. Users may also be allowed to configure some transformations and visualizations, or change values of some cross-filters (but the configuration capabilities may be limited, if compared to the designer’s capabilities).

## Recommended Data Formats

Since galaxy components can be created by different parties, those components may expect data to be in different data formats. If we have data in one format, but certain galaxy components expect data to be in other formats, data converters are needed. If there are many heterogeneous data formats, the number of required converters will increase. The cost for this consists of developers’ time needed to create the converters as well as the time to execute the converters each time the galaxy is run. In order to minimize this cost, it is strongly advised for most transformations to work with the recommended data formats (see below), where it is possible (although the Data Galaxy tool does impose any restrictions to the data formats).

The recommended data formats are as follows:

* For tabular data: CSV (Comma-separated values) using Excel syntax.  
  Excel syntax means that:
  + the comma is used a delimiter;
  + if some value contains commas, that value is decorated with quotes;
  + if there are quotes inside the value, the quotes are duplicated;
  + the first row of the CSV file must contain the titles of the columns.

If you use Apache Commons library for dealing with CSV in Java, use org.apache.commons.csv.CSVFormat.EXCEL for the CSV file format.

* For tree-like data: JSON, where the child nodes are put into a JSON array. If the given JSON is already an array, it is considered to be a forest (=a tree with multiple roots). Otherwise, the JSON object itself is a root for the tree. Given a node of the tree, the first attribute, whose value is a JSON array, is considered to contain the children of that node.
* For graphs: JSON with attributes “nodes” and “edges” storing JSON arrays. Each node must have the id attribute; each edge must have attributes “source” and “target”, whose values must be equal to the corresponding edge end points.
* For semantic web data: OWL functional-style syntax is preferred.

The abovementioned formats were selected for the following reasons:

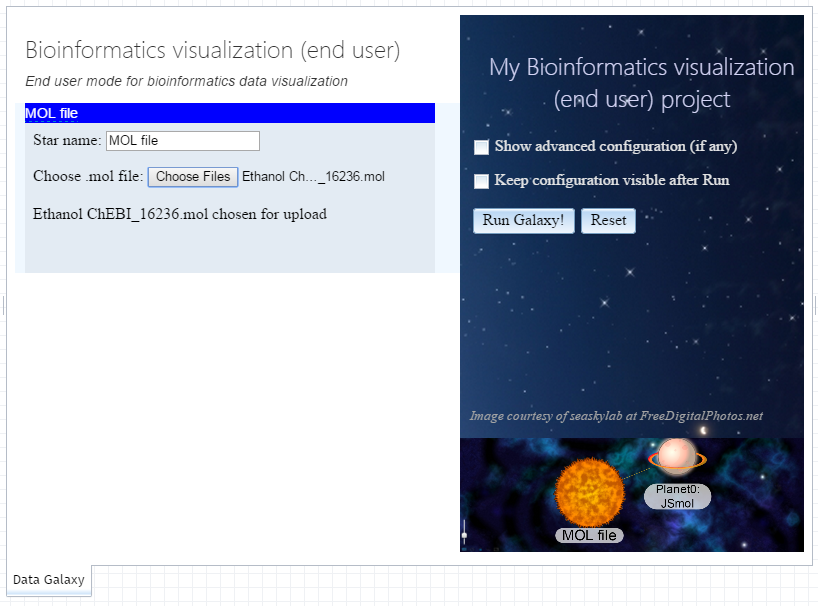
* The file size is being kept as small as possible, and that saves space for data stars.
* It is easy to create efficient data transformations (stellar winds) in different programming languages (the formats are simple, and many libraries for parsing them are available).
* It is easy to generate files in the given formats without any libraries.

The preferred encoding for all the files is UTF-8, since it is UNICODE and is compatible with ASCII-7.

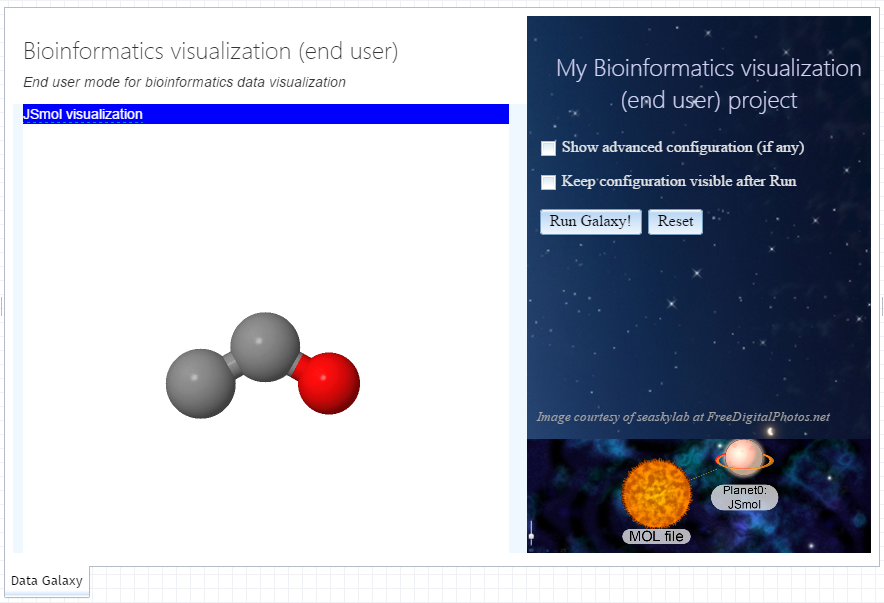
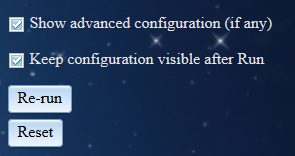
The preferred date format is YYYY-MM-DD, since it can be easily sorted lexicographically.

# The End-User’s Mode

When the user opens/creates a data galaxy project in the end-user mode, configuration frames for all the components that must be configured are displayed. For example, these may be the frames, where the user has to specify source data files. If the designer allowed the user to configure other components, their configuration frames can be made visible by selecting the “Show advanced configuration” checkbox.



After configuring galaxy components, the user clicks the “Run Galaxy!” button, and the galaxy is being executed. As a result, certain visualizations (specified by the designer) are shown.



If the user selects the “Keep configuration visible after Run” checkbox, the configuration remains visible, so the user can adjust some settings and re-execute the galaxy by clicking the “Re-run” button.

To start from scratch, the user can click the “Reset” button. In this case, the configuration specified by the user earlier is erased.

When the galaxy is being opened in the end-user mode for the first time, some adjustments can be made to the look-and-feel of the project. For instances, the page title and subtitle can be changed. Also, the layout of configuration and visualization frames can be modified. Usually, this work is performed by the designer, who opens the project in the end-user mode for testing purposes. Then the designer also adjusts the look-and-feel and clicks either the “Save layout”, or “Save layout forever” button. In the latter case, the end user won’t be able to change the layout and the titles any more.

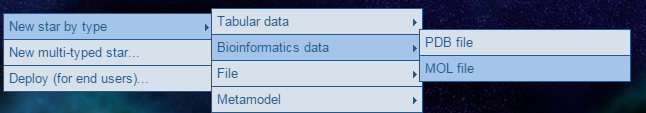
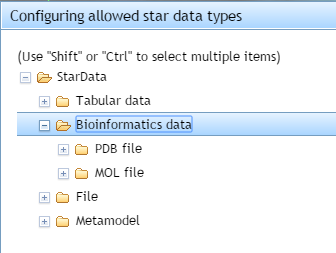
# The Designer’s Mode

A new galaxy project (see Section 1.1) contains an empty data galaxy. Then the designer adds galaxy components — stars (data sources or intermediate data), planets (data visualizations), stellar winds (data transformations), and cross filters.

## 3.1. Creating Stars (Data Sources)

First, the designer creates one or more source stars (data sources) that correspond to input files, data URLs, or other initial data. These data may be specified by the designer once for all end-users, or each end-user can choose their own data (the designer can choose the data as well, e.g., for testing purposes). The important thing is that data associated with each star are of some type. Thus, before specifying the data, a star data type must be specified. This can be done by the designer, or the designer may specify multiple acceptable types, and the end-user will have to choose one of the allowed types for the given star.

To create a single-typed star (for data of fixed format/type, e.g., CSV), right click somewhere in the galaxy and choose “New star by type”, and then navigate to the desired type (all types form a hierarchy that corresponds to the structure of the menu).



To create multi-typed star, choose “New multi-typed star”. In the dialog that appears choose data types that are allowed to be associated with the given star. You may choose a super type (e.g., “Bioinformatics data” to specify both “PDB file” and “MOL file”), which includes all subtypes. To select multiple types/supertypes, hold the “Shift” (for continuous selection) or “Ctrl” (for diverse selection) button. If you choose just one terminal type (i.e., not a supertype), then the star will become a single-type star.



Before a multi-typed star can be user, somebody has to choose a particular (final) star type. This can be done either by the designer, or by the end user depending on the use case. To clear a particular (final) star type, right click on the star and choose “Drop star configuration”. After that a particular (final) star type can be chosen once again.

To edit star configuration (e.g., to choose a file or other data source), right click the star and select “Configure…” (this configuration action is invoked automatically for newly created stars). A star-specific dialog will appear, where you can specify the data and their parameters (if any). To clear the data configuration, right click the star and select “Cleanup star data”. The cleanup process depends on the star data type (e.g., if a file was uploaded to the star during configuration, that file may be deleted).

## 3.2. Attaching Planets (Visualizations)

For each star (including the initial ones) a planet (visualization) can be attached. Since planets rely on the format of the data attached to the star, each planet can be attached only to stars containing data in the supported format. To see which planets can be attached to the given star, right click at the star and choose “New planet”. The submenu will list the available planets, and you can choose any of them (each planet can be attached multiple times, since the user may want to have the same visualization, but with different parameters).

To edit planet configuration (if applicable), right click the planet and select “Configure…” (this configuration action is invoked automatically for newly created planets). A planet-specific dialog will appear, where you can specify visualization parameters (if any).

To visualize the planet, right click the planet and select “Visualize…”. A dialog (or an external program – for some planets available in the desktop Data Galaxy tool) with a visualization will appear. If the corresponding star is not an initial star, then stellar winds (transformations) that lead to that star are being executed first.

## 3.3. Adding Stellar Winds (Transformations) To Obtain New Stars

Data from stars can be transformed to obtain new data that will be associated with new stars. Transformations are performed by stellar winds. Each stellar wind is emitted from one or more source stars and creates exactly one target star. The supported data types of the source and target stars are specific to each particular stellar wind. While most stellar winds will have just one source, some may require two or more sources (e.g., a stellar wind for merging data from two stars).

To add a stellar wind, select the desired source star (or stars[[1]](#footnote-1)), right click, and choose “New stellar wind”. The submenu will list the available stellar winds that you can choose for the selected set of stars. Then a configuration dialog will appear, where you can configure the stellar wind and (usually) the target star.

The stellar wind and the target star can be configured also by selecting “Configure…” from their context menu.

## 3.4. Adding Cross Filters

Cross filters can be attached to stellar winds (transformations) and planets (visualizations). To attach a new cross filter, right click on a stellar wind or a planet and choose “New cross filter” and then the desired filter. Only filters that are supported by the given stellar wind/planet are displayed. To attach an existing cross filter (that is already used by some planet or stellar wind), right click on a stellar wind or a planet and choose “Attach existing cross filter” and then the desired filter. Only supported and existing filters are displayed.

## 3.5. Editing the Galaxy

The galaxy designer can add and remove galaxy components, and reconfigure them. The designer can visualize planets (which causes certain galaxy steps to be executed) and execute certain stellar winds manually (selecting “Emit…” from the context menu of stellar winds). When the designer performs some actions on some galaxy components, other components that are affected may change their state. The designer can see the state of each galaxy component by looking at the component icon. The states are as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| State name | Star | Planet | St. wind | Cross filter | Description |
| CONFIGURATION\_UNKNOWN | √ |  |  |  | A multi-typed star does not have a particular type specified |
| CONFIGURATION\_OK | √ | √ | √ | √ | The configuration of the given component is correct |
| CONFIGURATION\_ERROR | √ | √ | √ | √ | The configuration of the given component is not complete or is incorrect |
| RUNNING | √ | √ | √ |  | The given component is being initialized or executed |
| RUN\_OK | √ | √ | √ |  | The given component has been successfully initialized or executed |
| RUN\_ERROR | √ | √ | √ |  | There was an error while initializing or executing the component |

When an error occurs, a galaxy component may provide an error message. To get this message, move the mouse pointer to the component, and a tooltip with a message will appear.

## 3.6. Data Pre-Conversion and Post-Conversion

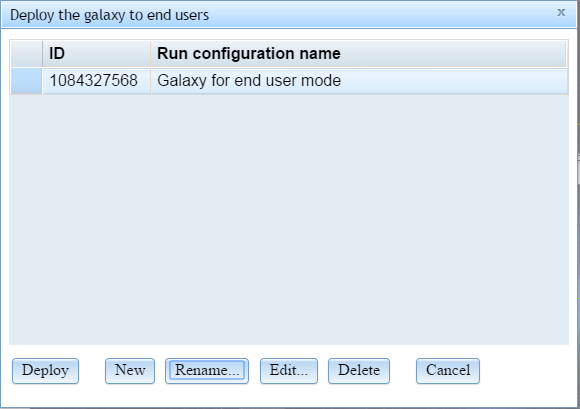
Although stars in Data Galaxies may contain data from different locations and in different formats, it is advised to convert all data to the recommended formats (see Section 1.3) before applying transformations (stellar winds). In the end, the data can be converted to the desired target format.

Certain transformations (stellar winds) for pre- and post-conversion are already available:

* Spreadsheet\_to\_CSV – converts any one Excel sheet (from .xls, .xlsx, or .xlsm file) into a CSV file.
* CSV\_to\_JSON – converts CSV to a JSON array of JSON objects.
* JSON array to tree – converts a JSON array to a JSON tree (the user has to specify the isParentOf function for determining the parent-child relationship between elements in the source array).

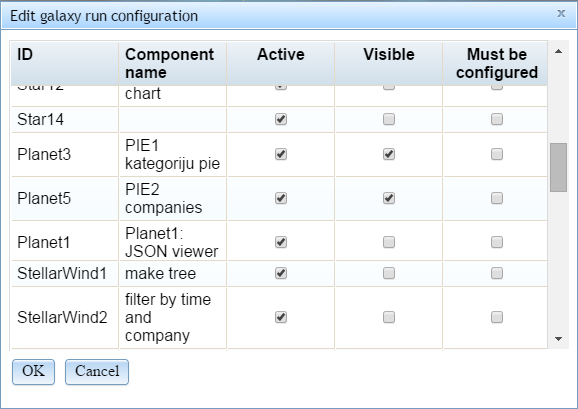
## 3.7. Deploying Data Galaxies for End-Users

When the designer has finished designing the galaxy, the galaxy needs to be deployed as a template for end users. To deploy the galaxy, right click somewhere on the galaxy and choose “Deploy (for end users)…”. A dialog window will appear.



In the dialog multiple running configurations can be specified. Each of them can contain different parameters and be deployed independently. To add a new configuration, click the “New” button and specify the name of the configuration. This name will also be the name of the template used by end users.

After that, the second dialog will appear (it can be launched later by clicking the “Edit…” button).



The dialog contains the list of all galaxy components. By default all components are active. The designer may switch off any component by unchecking the “Active” checkbox. In this case the component will be deleted from the galaxy (useful for some filters and planets that are needed only during the design time).

To make a component visible to the end user, the “Visible” checkbox must be checked. For stars, stellar winds, and cross filters, “visible” means “allow to configure”; for planets “visible” means “allow to configure and visualize”.

If some component requires input from the user, the “Must be configure” checkbox must be selected. This is useful, for example, when the user has to specify the source data for some star.

When the configuration is finished, click “OK”.

To deploy the selected configuration (from the first dialog), click the “Deploy” button.

In the web tool as well as in the desktop tool, the “Deploy” button automatically puts the deployed galaxy to the templates folder, thus, it will be available to the end users. However, if the deployed galaxy must be transferred to another computer, the deployed template must be copied manually (into the tools/DataGalaxy/templates folder).

If the template has been deployed in the desktop tool, it can be used also in the web tool, but then the template must be copied to the server (into the tools/DataGalaxy/templates folder).

# Developing New Galaxy Components

First, we mention the four steps that have to be performed, when developing a new galaxy component. Then we provide some details that are needed during those four steps.

## The Four Steps

To develop a new galaxy component type, the following four steps have to be performed:

1. Create a subdirectory in the tools/DataGalaxy/galactictypes folder. The name of the directory corresponds to the name of the galaxy component.
2. Create a metamodel file (<ComponentName>.mmd or <ComponentName>.ecore) that describes how the galaxy component has to be stored in the model repository as well as the component location in the component hierarchy. There are four supertypes for galaxy component types: StarData[[2]](#footnote-2), Planet, StellarWind, and CrossFilter. Other components are direct or indirect subtypes or these four types.  
   The metamodel file must contain a class with the name equal to the component name. The metamodel file must also contain a generalization relationship between this class and some supertype (that leads to one of the four types above).  
   The metamodel file may define also additional attributes and associations that are needed for the galaxy component.
3. Create a properties file (<ComponentName>.properties), which describes how to perform certain operations on the galactic type (e.g., how to visualize a planet or how to emit a stellar wind transformation). The properties file also specifies certain constrains that have to be met, when using the component (e.g., the source and target star types for a stellar wind). The format of the properties file is described in Section 4.2.
4. Implement the required operations for your component in a programming language supported by webAppOS web processors. As a special case, operations can be implemented using HTML/CSS/JavaScript (HTML files are placed into the web-root/galactictypes folder). More details are provides in Section 4.3.

Example.

1. A folder bin/galactictypes/YourStellarWind is created.
2. A metamodel file YourStellarWind.mmd with the following content is placed in the folder just created:  
   MMDefStart;  
     class YourStellarWind;  
     rel YourStellarWind.subClassOf.StellarWind;  
     attr YourStellarWind.specificAttribute:string;  
   MMDefEnd;
3. The properties file can be as follows (see details about the format in Section 4.2):  
   # The two properties below have values corresponding to webAppOS web calls.   
   # A predefined function configureViaHTML in the DataGalaxy Java class is used to  
   # show HTML files for configuration; HTML files are searched as  
   # tools/DataGalaxy/web-root-galactictypes/YourStellarWind/endUserConfiguration.\*.html  
   # and tools/DataGalaxy/web-root-galactictypes/YourStellarWind/designerConfiguration.\*.html  
   # the asterisk “\*” denotes where the desired frame dimensions are specified, e.g.,  
   # designerConfiguration.800x600.html.  
   designerConfigurationURI=staticjava:DataGalaxy#configureViaHTML   
   endUserConfigurationURI=staticjava:DataGalaxy#configureViaHTML  
     
   # You may create your own Java function to implement certain actions; these function also  
   # have to be specified as webAppOS web calls.  
   # For static Java functions specify the “staticjava:” prefix followed by the  
   # full class name (including packaged) followed by “#” followed by the static function name.  
   emissionURI=staticjava:galactictypes.YourStellarWind.YourStellarWindClass#youStaticJavaEmissionFunction  
     
   # a list of sets of allowed source star types; if a set contains two types, two stars of the corresponding  
   # types must be selected to match; supertypes may also be specified  
   allowedSourceTypeSets={XLS file,JSON file},{XLSX file,CSV}  
     
   # a list of names of possible target star types; supertypes may also be specified  
   allowedTargetTypes=CSV,Excel file  
     
   # thus, the given stellar wind matches any of the following declarations:  
   # YourStellarWind(XLS file,JSON file) => CSV  
   # YourStellarWind(XLS file,JSON file) => Excel file  
   # YourStellarWind(XLSX file,CSV) => CSV  
   # YourStellarWind(XLSX file,CSV) => Excel file  
     
   # known filter types (the filters may be attached and may not)  
   knownCrossFilterTypes=Date filter
4. Implement the necessary operations. In the case of stellar wind, these are:
   1. Operation to configure the stellar wind in the designer mode (a HTML file, e.g., tools/DataGalaxy/web-root-galactictypes/YourStellarWind/designerConfiguration.\*.html, must be provided).  
      Example:  
      Stellar wind name: <input type="edit" id="windName" value="New stellar wind"/>  
      Specific attribute value: <input type="edit" id="attrVal" value="Specific value"/>  
      <script>  
      function load(wind)  
      {  
       if (wind.name)  
       document.getElementById("windName").value = wind.name;  
       if (wind.specificAttribute)  
       document.getElementById("attrVal").value = wind.specificAttribute;  
      }  
      function store(wind)  
      {  
       wind.name = document.getElementById("windName").value;  
       if (!wind.name) {  
       wind.state = "CONFIGURATION\_ERROR";  
       wind.stateMessage = "The name of the stellar wind is not specified.";  
       }  
       wind.specificAttribute = document.getElementById("attrVal").value;  
      }  
      </script>

Notice that the stellar wind object (wind) is loaded from and stored in the repository automatically, since we use the predefined function configureViaHTML (we specified it in the .properties file as “staticjava:DataGalaxy#configureViaHTML”). If you want to use different way of configuration, you have to use some mechanism to access webAppOS web memory to work with the stellar wind object in the model repository.  
Notice also that in case of an error, the *state* attribute is set to denote the error, and the *stateMessage* attribute is also set to specify the error message.

* 1. Operation to configure the stellar wind in the end-user mode (a HTML file, e.g., tools/DataGalaxy/web-root-galactictypes/YourStellarWind/endUserConfiguration.\*.html, must be provided).  
     Example:  
     Stellar wind name: <input type="edit" id="windName" value="New stellar wind"/>  
     <script>  
     function load(wind)  
     {  
      if (wind.name)  
      document.getElementById("windName").value = wind.name;  
     }  
     function store(wind)  
     {  
      wind.name = document.getElementById("windName").value;  
      if (!wind.name) {  
      wind.state = "CONFIGURATION\_ERROR";  
      wind.stateMessage = "The name of the stellar wind is not specified.";  
      }  
     }  
     </script>

Notice that the end-user mode HTML file does not allow the user to enter the value for *specificAttribute*. This may be useful, if the galaxy component must provide more configuration capabilities to the designer and less to the end user.

* 1. Operation to emit the stellar wind. Since we specified our own Java function name in the .properties file, we need to implement that function. The header of the function is as follows (*webmem* is a pointer to webAppOS web memory, which is used to access the model repository; *r* is a reference to the EmitStellarWind instance):  
     package galactictypes.YourStellarWind;  
     …  
     public class YourStellarWindClass {  
      public static boolean emission(IWebMemory webmem, long r) {  
      …  
      }  
     }

Since we have specified that the stellar wind accepts “Date filter”, the emission function must check, whether one or more date filters are attached, and, if yes, take their values into a consideration.

## The Format of the Properties File

The properties that have to be specified in the .properties file depend on the type of the galaxy component (star data, stellar wind, planet, or cross-filter). The following table lists possible properties, to which components they apply, and what values are expected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Property name | Star | Planet | St. wind | Cross filter | Description |
| designerConfigurationURI | √ | √ | √ | √ | A webAppOS web call that will be used to configure the given galaxy component in the designer’s mode. An instance of ConfigureStar, ConfigurePlanet, ConfigureStellarWind, or ConfigureCrossFilter will be passed to the transformation. |
| endUserConfigurationURI | √ | √ | √ | √ | A webAppOS web call that will be used to configure the given galaxy component in the end-user’s mode. An instance of ConfigureStar, ConfigurePlanet, ConfigureStellarWind, or ConfigureCrossFilter will be passed to the transformation. |
| initializationURI | √ |  |  |  | A webAppOS web call that will be used to initialize the corresponding star data (i.e., open a file, or mount some external model repository). An instance of InitializeStar will be passed to the transformation. |
| finalizationURI | √ |  |  |  | A webAppOS web call that will be used to finalize the corresponding star data (i.e., close a file, or unmount some external model repository). An instance of FinalizeStar will be passed to the transformation. |
| cleanupURI | √ |  |  |  | A webAppOS web call that will be used to delete the corresponding star data or to disconnect from star data. An instance of CleanupStar will be passed to the transformation. |
| designerVisualizationURI |  | √ |  |  | A webAppOS web call that will be used to visualize the corresponding planet in the designer’s mode. An instance of VisualizePlanet will be passed to the transformation. |
| endUserVisualizationURI |  | √ |  |  | A webAppOS web call that will be used to visualize the corresponding planet in the end-user’s mode. An instance of VisualizePlanet will be passed to the transformation. |
| allowedStarDataTypes |  | √ |  |  | One or more star data type names delimited by commas. These names specify star data types, to which the given planet can be attached, i.e., which data types this planet is able to visualize.  Example:  allowedStarDataTypes=Metamodel,Ontology |
| knownCrossFilterTypes |  | √ | √ |  | One or more cross-filter type names delimited by commas. These names specify filter types this planet (stellar wind) is able to take into a consideration, when visualizing (transforming) data.  Example:  knownCrossFilterTypes=BoundsFilter |
| emissionURI |  |  | √ |  | A webAppOS web call that will be used to execute the corresponding stellar wind (data transformation). An instance of EmitStellarWind will be passed to the transformation. |
| allowedSourceTypeSets |  |  | √ |  | One or more sets of star data type names. Each set specifies which star data types can be used as sources for this stellar wind. Sets are specified in “{” and “}”, and are delimited by commas.  Example:  allowedSourceTypes={CSV,MMDFile},{ECore file},{Metamodel} |
| allowedTargetTypes |  |  | √ |  | One or more star data type names delimited by commas. These names specify star data types this stellar wind is able to produce.  Example:  allowedTargetTypes=Metamodel,Ontology |

## Using HTML to Implement Galaxy Components

To implement configuration, visualization, emission actions in a language other than HTML/JavaScript, you have to define and specify a webAppOS web call for that.

For HTML/JavaScript implementation there are predefined Java functions that perform certain work that otherwise had to be implemented manually (including loading and storing the galaxy component from the repository and transferring it to/from the client browser). These functions are:

* staticjava:DataGalaxy#configureViaHTML
* staticjava:DataGalaxy#visualizeViaHTML
* staticjava:DataGalaxy#emitViaHTML

The configureViaHTML function searches for the file designerConfiguration.html, endUserConfiguration.html, designerConfiguration.\*.html, or endUserConfiguration.\*.html (the asterisk denotes the place for default frame dimensions, e.g., “800x600”) and displays it in the data galaxy. Also, it invokes the load(component) function, which has to load the values from the component object, from that HTML. When the component needs to be saved into the repository, the store(component) function, which has to save values into the component object, is called. If the component object has certain links in the repository that have to be traversed, the load/store functions may use webAppOS client-side webmem object to access the repository.

For asynchronous calls you may specify functions loadAsync(component, fAfter) and storeAsync(component, fAfter) instead of synchronous load and store. In this case, call the provided fAfter() at the end of each function.

The visualizeViaHTML function searches for the file designerVisualization.html, endUserVisualization.html, designerVisualization.\*.html, or endUserVisualization.\*.html (the asterisk denotes the place for default frame dimensions, e.g., “800x600”) and displays it in the data galaxy. Also, it invokes the load(component) function, which has to load the values from the component object, from that HTML. The load function may be either synchronous, or asynchronous.

The emitViaHTML function searches for the file emission.html or emission.\*.html (the asterisk denotes the place for default frame dimensions, e.g., “800x600”), displays it in a temporary frame in the data galaxy, and invokes the synchronous emit(component) function or asynchronous emitAsync(component, fAfter) function. In addition, you may specify the finalize() function that will be called, when the user wants to force terminating of the stellar wind execution. Do not forget to call fAfter specified in the emitAsync (in case of asynchronous emission implementation).

## Additional Guidelines

* For configuring components it is advised to use HTML/JavaScript, since it can be used both in web and desktop versions of the Data Galaxy tool. Also, HTML configurations can use webAppOS webmem object. In case of HTML/Javascript the galaxy also performs the forward cleanup of the changed component (since, when a configuration of some galaxy component changes, all further stars created by stellar winds must be cleared).
* Stellar winds must ensure that their target star data type has been inserted into the repository. You can use the webAppOS web call “putGalacticTypeIntoMetamodel” for that, e.g.,  
  webappos.webcall("putGalacticTypeIntoMetamodel", "JSON file");

Also, do not forget to create the target star and specify the default configuration for it (look at the CSV\_to\_JSON stellar wind implementation).

* If the component object in the repository is empty or lacks some attribute values or links, assume that the default values have to be used. This simplifies the implementation of the cleanup/reset functionality in data galaxies.
* Where the user has to specify some index (e.g., column number) during component configuration, indexes should start from 1.
* Do not forget to set the component’s state value (e.g., to “CONFIGURATION\_ERROR” or “RUN\_ERROR”) in case of some error, and the stateMessage value to the error description.
* In HTML/JavaScript code you can use useful functions from webappos.js.
* When implementing certain functions specified in the properties file manually (i.e., not in HTML/JavaScript), always launch RefreshGalaxyCommand (via webmem.submit). In case of error, set the corresponding values to the state and stateMessage attributes.
* When implementing certain functions specified in the properties file manually (i.e., not in HTML/JavaScript), you may need to cleanup certain parts of data galaxy (i.e., if the source file name changes during configuration, all forward stars (stellar wind results) must be cleared).
* When configuring data stars, you may have to discern, whether the star is an initial star or a star resulting from some stellar wind. I.e., the initial star may require to select the file, while non-initial stars may just require a file name (this name can be used to create resulting star data). Or, you can use hardcoded names for non-initial stars.

### Injection points

The table below describes the injection points, where you could place your code.

|  |  |
| --- | --- |
| Injection point (in the file system) | Description |
| **bin/**  **galactictypes/*YourType***  (base injection point) | A folder containing:   * the metamodel of your galatic type (.mmd or .ecore file), * the configuration file (.properties file) describing meta-information for your galactic type as well as how to execute galaxy commands related to your component, * (optionally) the server-side code of your component (e.g., Java classes located in the Java package galactictypes.*YourType*). |
| **web-root/galactictypes/**  ***YourType***  (web-root injection point) | A folder containing HTML files (and, perhaps, additional .js and .css files) that are displayed, when your galaxy component is being configured, visualized (for planets), or executed (for stellar winds). HTML files for both designer mode and end-user mode are placed here. Unlike the base injection point, all the files in the web-root injection point are accessible from the client browser (using the path “/galactictypes/YourType”). |

1. To select multiple stars, hold the “Shift” or “Ctrl” button. [↑](#footnote-ref-1)
2. Notice that instead of the Star type, we have the StarData type. That is done intentionally, since the same multi-typed star may be associated with data of different types, thus, the star object remains the same, but the StarData object changes. [↑](#footnote-ref-2)