Greybox Developer Guide

Introduction

The Greybox UI is a browser based application which visualizes factory ‘Runplan’ data and the factory’s performance indicators based on that ‘Runplan’. This UI component can be used in a standalone fashion or integrated as part of a wider solution.

Technical requirements

Greybox is developed in HTML[1] and JavaScript[2] and utilizes several open source[3] libraries such as jQuery[4]. It is compatible on the Chrome[5] browser but can be adapted to work an all mainstream browsers. It can be used as a demonstration tool offline, using static data provided in the form of JSON objects, or used online in conjunction with a database via a web service. For the latter, it requires a web server to service incoming web requests (Apache HTTP Web Server[6] 2.2.x is the reference Web Server for Greybox UI).

Reference Architecture Components (for online mode)

* Windows Server 2003 SP2
* Apache HTTP Server 2.2.x
* PHP 5.3.X[7]

Note: The above components are recommended on the basis that the UI was developed and tested with these components. Other components and platforms such as Linux are not explicitly permitted and can be used under the proviso the developer understands the fundamental differences in relation to the Greybox UI as described in this document.

UI Component Description

/planning.html – HTML code for planning screen

/index.html – Redirects to planning.html

/css/ - contains stylesheets for all components

/data/ - contains definitions of included sample JSON objects (see Data section below)

/img/ - contains all images used in the UI

/js/ - contains all Javascript code

/js/3rdparty/ – contains all third party Javascript code (e.g. jQuery)

/js/greybox /– contains all greybox JS code

/js/greybox/data.js – contains methods for calling CGI script/web service to populate data structures

/js/greybox/common.js – contains common variables and helper methods used by the rest of the code

/js/greybox/planning/ - contains all code for the planning screen

/js/greybox/planning/toolbar.js – contains code pertaining to the ribbon toolbar

/js/greybox/planning/runplan.js – contains most of the code pertaining to drawing the main interactive runplan portion of the planning screen

/js/greybox/planning/runplan-context-menu.js – contains the code for handling the right mouse button menu on the runplan

/js/greybox/planning/planning.js – code pertaining to the planning screen itself

/js/greybox/planning/planning-time-line.js – code that handles rendering the time line on the planning screen

/js/greybox/planning/planning-tabs.js – used for tab handling – e.g. adding new tabs

/js/greybox/planning/planning-rules.js – contains logic for creating and modifying rules and updating the rules right hand panel

/js/greybox/planning/planning-rules-wizard.js – contains code for the rules wizard (incomplete)

/js/greybox/planning/planning-product-panel.js – contains code for rendering the product panel in the right hand side menu

/js/greybox/planning/planning-product-config.js – module for handling the product configuration popup dialog box.

/js/greybox/planning/planning-ppi-panel.js – handles PPI updates and updating the PPI panel in the right hand menu

/js/greybox/planning/planning-feedback.js – logic for the feedback panel

/js/greybox/planning/planning-common.js – contains methods that are common to all planning screen functions

Data

Greybox UI has two possible sources of data input: predefined JSON objects either located in the runplan-opt.js file in the ‘/data’ directory or in planning.js in ‘/js/greybox/planning’, and data obtained from an external source via JavaScript AJAX requests.

JSON Object Structure

This section describes the relevant JSON object structures that the UI uses for rendering and PPI calculations. These can be modified in the JS source files or pulled down via web request.

‘energy’ – The energy requirements of each tool state. A reference implementation is provided that sets the energy usage for Idle and PowerSave. The JSON format of the object is as follows:

{ “<state>”: {“energy”: <energy usage>}, …};

‘outputRequired’ – The product line and various production targets of each product. The format is as follows:

{“products”: [{“product”:”<product name>”, “priority”: “<priority>”, "outputTargetTarget":"<production target>","processed":"<production already achieved>","remain":"<remainder>","offset":"<offset>","ulimit":"<upper production limit>","llimit":"<lower production limit>","meet":"<minimum target to meet”","targetType":"<which target we are aiming for (llimit, meet or ulimit)>"}, … ]};

‘runPlanData’ – The main runplan and the positioning of the task blocks. Summary of the output is as follows:

{ “runplan”: [{“time”:<task start time>, “toolid”:”<tool id>”, “state”:”<task state name>”, “setup”:”<product this state refers to>”}, …]};

‘rules’ - A list of dynamic constraints that have been saved. Format as follows:

{ “rules”: [

{

"tools":”<list of tools this rule applies to>”,

"toolName":"<same as tools>”,

"numOfTool":"<number of tools”>,

"state":"<type of state rule applies to>”,

"product":"<product name rule applies to>”

"lot":"<lot id of task block that the rule applies to”

"target":"<unused>”,

"added":"<time in milliseconds when rule was added>”,

"duration":"<length rule will be active for>”,

"id":"<rule id>”,

"username":"<user who created the rule>”,

"windowFrom": “<nearest possible start time>”,

“windowTo”: “<furthest possible end time>”

},

…

]};

Online Mode

Greybox UI code contains AJAX calls which, when enabled, will attempt to communicate with a simple web service/CGI script to return JSON objects which are used to render the runplan instead of the static pre-defined JSON objects included with the code. This CGI script can be implemented by the user and constructed via whatever method they like (e.g. reading from a database) as long as it conforms to the expected data structures as described above. The reference implementation used PHP but any technology may be used as long as the output describes a valid JSON object.

Turning on online mode & implementing operations

In order to activate calling out to a web server, the variable ‘remoteEnabled’ in the common.js directory must be set to true. In addition, the ‘kapPhpFile’ should be set to the path of the web service target. This does not necessarily need to point to a PHP script - any web-enabled language will work.

The web service requests are sent to the location as described in the kapPhpFile. Data is passed through using HTTP GET parameters and, in one case, additional HTTP POST data. Every request contains an ‘op’ GET parameter – this defines the type of operation. The other parameters passed in and the data returned depend on which operation it refers to. The following is a list of operations that will be called in online mode: these will all need to be implemented to ensure the application runs successfully. The JavaScript functions that calls these operations are contained in ‘/js/greybox/data.js’ and can be modified if necessary.

Op: servertime

Params: none

Returns: current time on server in milliseconds

Op: createsession

Params: id – a string containing a unique session id

Returns: none

Notes: This is used for multi-tab support and is called any time a new tab is created. Use this to set up anything you may need to support multiple user sessions

Op: destroysession

Params: id – a string containing a unique session id

Returns: none

Notes: This is used for multi-tab support and is called any time a tab is closed. Use this to set up anything you may need to clean up a tab session.

Op: destroyAllUserSessions

Params: None

Returns: None

Notes: This is called when all user sessions are to be destroyed, e.g. on browser close

Op: comb

Notes: Returns a runplan representation, including historic data

Params: sessionId (optional) – a string containing a unique session id, used for multi-tab operation

start – specify the time at which to start the runplan. Format “YYYY-MM-DD HH:MM:SS.MS”

Returns: a ‘runPlanData’ object representing the runplan

Op: cur

This is the same as comb but with no historic output.

Op: hist

Notes: Returns a runplan representation of a range of dates

Params: histFrom/histTo – strings containing the start and end time of the data range. Format “YYYY-MM-DD HH:MM:SS.MS”

Returns: a ‘runPlanData’ object representing the runplan

Op: outputRequired

Notes: Returns a list of products and associated production data

Params: week – week number of required product data, either 1 or 2 (1 = current week, 2 = next week)

Returns: an ‘outputRequired object representing the product data

Op: histoutputRequired

Notes: Returns a list of products and associated production data from historic data

Params: date – what date the product information correlates to

Returns: an ‘outputRequired’ object representing the product data

Op: energy

Notes: Used to retrieve state energy information

Params: none

Returns: an ‘energy’ object representing the energy data

Op: userinfo

Notes: Used to retrieve username of person using UI

Params: none

Returns: a plain string containing the username

Op: listrules

Notes: Used to retrieve a list of current dynamic constraints

Params: none

Returns: a ‘rules’ object

Op: feedback

Notes: Used to send feedback from feedback form to server for storing

Params: content – feedback text

Name – value of user inputted name field

Returns: None

Op: updateoffset

Notes: Used to update a product’s offset

Params: sessionId – session Id corresponding to the tabbed session

Offset – value to set offset to

Product – product name to change

Returns: None

Op: dellalldc

Notes: Used to delete all dynamic constraints

Params: sessionId – sessionId – session Id corresponding to the tabbed session

Returns: None

Op: outputTarget

Notes: Used to store a runplan on the server

Params: id – session id

POST data - JSON representation of runplan using ‘runplanData’ object

Returns: None

Op: adddc

Notes: Used to add a dynamic constraint

Params: Tool – Tool name(s) rule applies to

Num – Number of tools in rule

State – State relating to rule

Duration – Length of rule

WindowFrom – Start period of when rule can be placed

WindowTo – End period of when rule can be placed

Added – when rule was added (milliseconds)

sessionId – session Id relating to current tab

Product - product name rule relates to

lotId – lot id rule relates to

dcId – id of dynamic constraint

type – rule type (0 = resize, 1 = drag, 2 = insert, 3 = change, 4 = delete)

Returns: None

Op: tools

Params: none

Returns: A JSON object representing the current set of all tools (see: JSON object structure section)

# References

[1] HTML – Hyper Text Mark-up Language. <http://www.w3.org/html/>

[2] Javascript -JavaScript (JS) is an interpreted programming language. <http://en.wikipedia.org/wiki/JavaScript>

[4] JQuery – A javascript library. <http://jquery.com/>

[5] Chrome Browser – Browser owned by Google <https://www.google.com/intl/en/chrome/browser/>

[6] Apache Web Server – HTTP Web server - <http://httpd.apache.org/>

[7] PHP – Hypertext Pre-processor. <http://php.net/>