Stack Control

ARCHIBUS uses Stack Control in a handful of marquee views. You do not need a `<panel />` tag to initialize Stack Control: you can initialize it from JavaScript in all cases.

To include stack control in a view, use the file ab-common-controls-stack.axvw:

```
<panel type="view" id="stackControl" file="ab-common-controls-stack.axvw"/>
```

The common controls file contains necessary stack control resources: JavaScript, CSS, localized strings, and third-party libraries.

Stack Control's data is driven from datasources in these files:

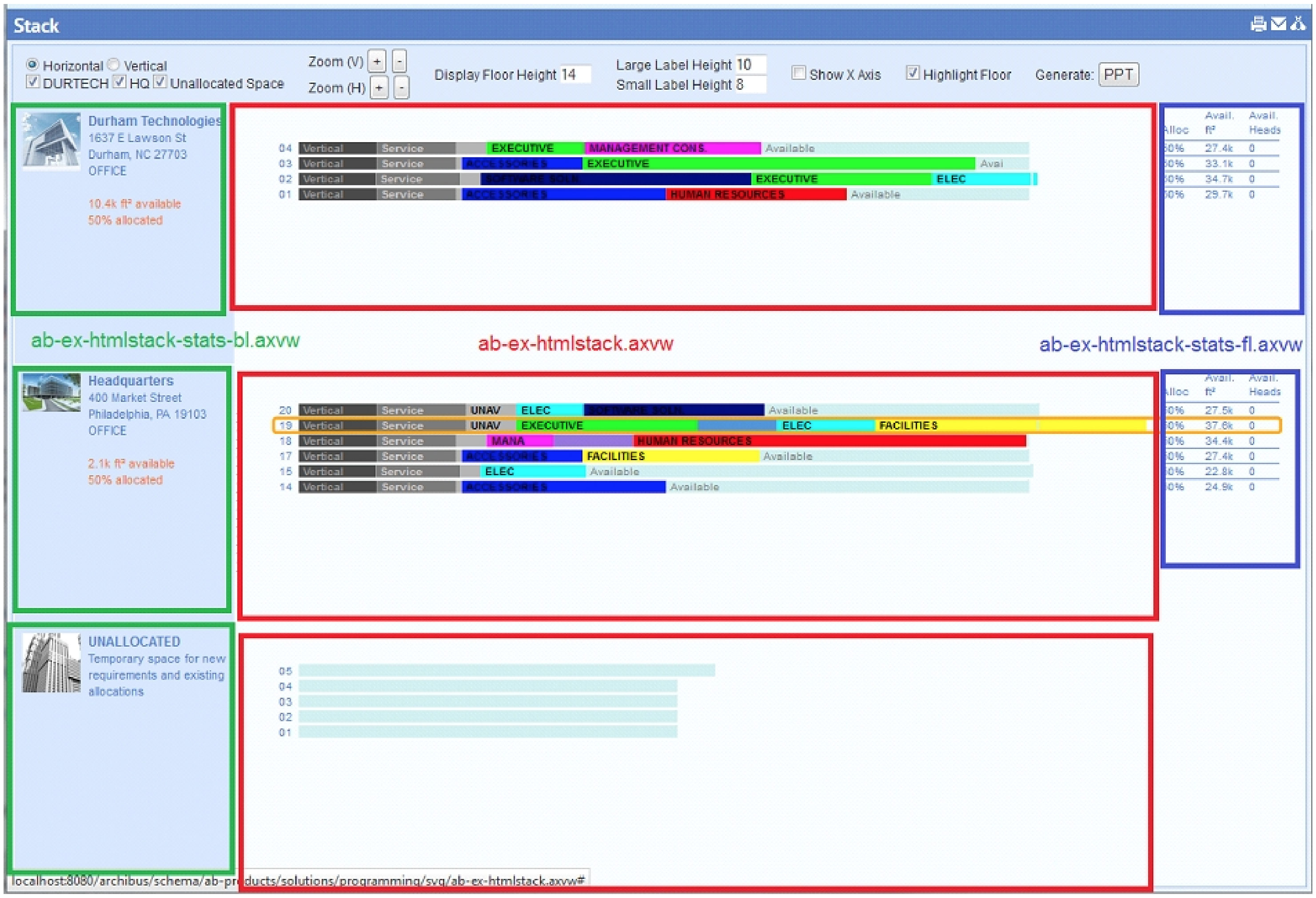
File	Holds	Represented in the below image by color
ab-ex-htmlstack-stats-bl.axvw	the building statistics	green
ab-ex-htmlstack.axvw	the stack elements in the actual stack chart	red
ab-ex-htmlstack-stats-fl.axvw	the floor statistics	blue

All three files are located in this folder:

...\schema\ab-products\solutions\programming\stack

Stack Control uses the same datasources for each building, adding restrictions for the relevant bl_id, portfolio_scenario_id, and asOfDate to restrict the records.

The image below uses color codes green, red, and blue to show the datasource for each part of the stack plan:



You can find this view in:

...\schema\ab-products\solutions\programming\stack\ab-ex-htmlstack.axvw

Runtime API

Use the runtime API to initialize stack control, and to use this feature wherever it is available.

Initialization

Use the following constructor to initialize stack control.

```
var stack = new Ab.stack.HtmlStack(divId, config);
```

Where:

`divId` - a string that holds the id of <div/> for the stack (for example, “stackContainer”).

`config` - an Ab.view.ConfigObject object that holds parameters that control the stack. (See Configuration object below.)

Add a building to the stack

The stack starts empty. During a working session, planners add selected buildings to the building list. The stack displays the buildings in the list, using data in groupDatasource. To remove a building from the display, remove it from the building list. Use the third method below to clear all buildings from the list.

```
stack.addBuilding( "HQ") ;
stack.removeBuilding( "HQ") ;
stack.clearAllBuildings();
```

Currently selected building and floor

These methods return the Building Code, Floor Code, and Group Code for the selected building, floor, and group. Other console actions use these codes. These actions depend on the building and floor currently selected in the stack plan.

```
stack.getSelectedBuilding() ;
stack.getSelectedFloor() ;
stack.getSelectedGroup() ;
```

Refreshing the stack diagram

Some callback actions change the underlying group data for the stack. You can refresh the stack diagram with either of two methods:

```
stack.refresh()
```

```
stack.build()
```

If the underlying data for the stack diagram have changed, use `stack.refresh()`. This method queries the database, and then builds the stack diagram.

Some actions, such as zooming or enlarging the stack diagram, do not require refetching the data. In that case, simply use `stack.build()` to redraw the stack diagram.

Actions and event handlers

Event handlers may be specified as a property in the stack control's configuration object. See Configuration object below.

OnDrop

The drop action invokes a callback registered by the application, and passes an object to the callback. The object contains three properties:

- sourceRecord
- targetRecord
- previousRecord

These properties correspond to a database record for a particular stack element. When you drag and drop, you typically pick up an element (source), drag to a different location, and drop onto another element (target). Because we want to drop the source between two elements in the stack, the drop callback returns both the target and the element before the target (previous), which allows application developers to pinpoint the sourceRecord's new position in the sort order.

Note: You can retrieve the group code, building code, and other information from the sourceRecord, targetRecord, or previousRecord, to apply and customize required business logic.

The callback action, which is part of the application, updates the group's table record with the new location. The callback action may:

- Update the Building and Floor location of the dropped item in the database, to reflect the new allocation of the group as a whole.
- Split the group record into two records with different start and end dates, and different locations.

  This happens in the case of a move. If you move a group, the action ends the allocation at the old floor location and creates a new, duplicate group record with the allocation in the new location. In this way, the group table can track the location of the group over time.
- Update the Sort Order (gp.sort_order) of all groups on the floor to make the dropped group appear after the drop-target group,

  The stack does this so that planners can display groups with high affinity for each other, next to each other in the stack.
- Update the area of the drop-target group.

  May be required if you drop a negative group area representing a decrease in size for the current time period.
- Disoplay a pop-up a message.

  May be required if you drop a negative area for one department onto another department. The application does not take any action, and displays a pop-up message to say so.

The application's callback action makes changes to the group table.

The callback action then calls `refreshStack()` to display the change on the stack diagram.

Notice that you can drop a group on the "Unallocated" building. In that case, the Building Code and Floor Code supplied to the callback as the drop target location are blank.

onTooltip

The tooltip action invokes a callback registered by the application, and passes an object to this callback. The application uses this object to control and format the tooltip content. For example:

```
    onTooltip: function(item) {
        var str = "<strong>ID:</strong> " + item['gp.gp_id'] + "</span>";
                str += "<br/><strong>Building:</strong> " + item['gp.bl_id'] + "</span>";
                str += "<br/><strong>Floor:</strong> " + item['gp.fl_id'] + "</span>";
                str += "<br/><strong>Division:</strong> <span>" + item['gp.dv_id'] + "</span>";
                str += "<br/><strong>Type:</strong> <span>" + item['gp.allocation_type'] + "</span>";
                str += "<br/><strong>Order:</strong> " + item['gp.sort_order'] + "</span>";
                str += "<br/><strong>Color:</strong> " + item['dv.hpattern_acad'] + "</span>";
                str += "<br/><strong>Area:</strong> " + item['gp.area_manual'] + "</span>";
        return str;
    }
```

OnClick

When you click a group, stack control highlights the group, and sets that group's building and floor as the currently selected building and floor. Whenever you select a group, the `OnClick` action always updates the selected building and floor, regardless of the allocation type for that group.

Configuration object

This configuration object holds parameters that control the stack:

Parameter	Purpose
stackOrientation	*Horizontal* or *vertical*. Specifies how to arrange buildings in the stack diagram: horizontally -next to each other, or vertically - one on top of another.
displayFloorHeight	The floor height, in user-display units, used to draw the stack plan.
buildings	List of buildings to display in the stack, for example, ["HQ", "BOSMED"]. The stack diagram starts empty for every session. Use the function, Add Building to Stack, to add a building to the building list.
groupDatasourceView	The view or .axvw file that holds source data to support the stack diagram.
groupDatasource	The data source used to draw the floor outline, roof outline, and each stack group. groupDatasource would respond to queries to the gp table.
floorDatasourceView	The view or .axvw file that holds source data to support floor statistics.
floorDatasource	The data source used to create floor statistics.
buildingDatasourceView	The view or .axvw file that holds source data to support the building profile and building statistics.
buildingDatasource	The data source used to create the building profile and building statistics.
portfolioScenario	The gp.portfolio_scenario value used to restrict a query to the group table, to retrieve only groups for the current scenario. A sample portfolio scenario would be, "3434 - Marlborough Expansion".
asOfDate	The date for which to draw the stack diagram. Stack control passes this date to the query that returns group data, as groups have start and end dates for their allocation. The control also passes this date to the OnDrop and OnClick event callbacks, as the callback needs the date to update the group record according to the date the action occurred: for example, the date of a move or a lease signing.
clickHandler	Event handler for when the planner clicks on a group.
rightClickHandler	Event handler for when the planner right-clicks on a group.
tooltipHandler	Allows the application developer to specify contents of a tooltip.
dropHandler	Event handler for when a group is dropped onto a stack.

Group Datasource configuration object

Stack Control uses these parameters to interpret data from the group data source:

Parameter	Purpose
groupLabels	A list of fields to designate labels for each group in the stack, for example: "gp.planning_bu_id", "gp.dv_id;gp.dp_id", and "gp.dp_id;gp.gp_function". If the diagram contains one label field, place the label in the center. If the diagram contains more than one label field, place each label on its own line within the group. Make the last line smaller in height. It is easier to read the values if stack control wraps all lines the same way, than if it wraps some lines one way, and others another way based on the available width of the group.
areaField	The field that contains the area of the group, for example: "gp.area_manual".
sortOrderField	The field that controls the how the floor is sorted, for example: "gp.sort_order".
highlightField	The field that holds the highlight color, for example: "dp.hpattern_acad".
headcountField	The field that contains the total headcount of the group, for example: "gp.count_em".
allocationTypeField	The field that contains the allocation type, for example: "gp.allocation_type". See the enumerated values below this table.
buildingField	The field that contains the Building Code, for example: gp.bl_id.
floorField	The field that contains the Floor Code, for example: gp.fl_id.
floorNameField	The field that contains the Floor Name, for example: fl.name or "01 - Offices".
groupField	The field that contains the Group Code, for example: gp.gp_id. Used primarily to map unassigned colors.
asOfDate	There is no need to know the start and end date fields. Stack control passes asOfDate to the groupDatasource object, which paremeterizes the query using the asOfDate.

By using just the allocation_type and the Groups table, the program can:

- Draw the stack plan at a period in time.
- Calculate remaining available area and headcount.
- Calculate utilization statistics.

Group records that are part of a scenario have these Space Allocation Type values:

- Usable Area - Area that comes from Floors for owned spaces or Leases for let space.
- Allocated Area - Area that is allocated as in use by a bu, dv or dp.
- Unavailable Area - Vertical penetrations, service areas, floor areas not leased by organization, owned areas that are sublet.

The Space Allocation Type value is blank if the group is not part of a space planning scenario, for example, if the group is part of a group-level space inventory.

Calculate Remaining Available Area as follows:

> (Usable Area – Owned group areas + Usable Area – Leased group areas) - (Unavailable Area group areas) - (Allocated Area group areas) on the floor.

Do not subtract the Unavailable - Vertical Penetration Area, the Unavailable - Service Area areas, or the Unavailable – Remaining Area areas. Vertical Penetration, Service Area, and Unavailable Remaining are not part of the original usable area for the floor.

Calculate Available Headcount as follows:

> Total headcount for "usable" areas, minus total headcount for "allocated" areas.

Ignore records that have an allocation_type of "None". These records should not exist in a portfolio scenario.

Enumerated values:

- *NONE;*None;
- *Usable Area - Owned;*Usable Area - Owned;
- *Usable Area - Leased;*Usable Area - Leased;
- *Allocated Area;*Allocated Area;
- *Unavailable Area;*Unavailable Area;
- *Unavailable - Vertical Penetration Area;*Unavailable - Vertical Penetration Area;
- *Unavailable - Service Area;*Unavailable - Service Area;
- *Unavailable - Remaining Area;*Unavailable - Remaining Area;

The group query sorts all group records by the gp.sort_order,so as to display the Allocated Area groups in predictable order.

Cosmetic settings

Parameter	Purpose
labelLargeTextHeight	Height for first line of label text, for example: 10 px.
labelSmallTextHeight	Height for second line of label text, for example: 8 px.
areaColorUnavailableVerticalPenetration	Color of Vertical Penetration areas on the stack.
areaColorUnavailableSupportSpace	Color of Support Space areas on the stack.
areaColorUnavailableRemaining	Color of Remaining Unavailable areas on the stack.
areaColorUnavailable	Color of Unavailable areas on the stack.
areaColorAvailableOwned	Color of Available Owned areas on the stack.
areaColorAvailableLeased	Color of Available Leased areas on the stack.
areaColorAllocatedList	List of colors for use on allocated areas. If the hpattern_acad field has no value, stack control uses the next available color in this list.

groupDatasource

When you change the groupDatasource, the console uses the SUM and GROUP BY fields to summarize group data at four levels of detail:

- Business Unit - planning_bu_id
- Division - dv_id
- Department - dv_id+dp_id
- Functional Group - dv_id+dp_id+gp_function

Query results are identical, except that:

- Some query results include more fields used for the groupDatasourceLabels.
  For instance, a summary by Business Unit would not include the Division field, since that is not part of the GROUP BY or SUM fields.
- Different queries refer to different standard tables - for example, bu, dv, or dp - in order to retrieve the highlighted field data (hpattern_acad).

The query also refers to:

- The Floors table to retrieve Floor information: Floor name (fl.name), and Cost per Area (fl.cost_sqft).
- The Buildings table to retrieve Building information: Building Name (bl.name), Building Graphic (bl.bldg_photo), Construction Type (bl.construction_type), Address (address1, zip, city, state, country), Use (use1), Cost per Area (bl.cost_sqft), Market Value (value_market), Book Value (value_book).

Generating the stack diagram

List of buildings

The configuration object's *buildings* parameter holds all buildings to display. Display buildings in a horizontal or vertical fashion, based on the stackOrientation.

Unallocated building

If any groups in the planning scenario have no Building Code or Floor Code, add those groups to an Unallocated building at the end of the stack.

Always create an Unallocated building with at least one unoccupied floor, even if you have no Unallocated groups in the stack. Doing so lets the planner Unallocate a group in order to move it elsewhere.

Stacking the floors

The table below lists allocation types and default colors for space on each floor. For each floor, place allocation types from left to right in this order:

Allocation Type	Default Color and Label
Unavailable - Vertical Penetration Area	Dark gray, labeled with tooltip
Unavailable - Service Area	Medium gray, labeled with tooltip
Unavailable - Remaining Area	Light-medium gray, labeled with tooltip
Unavailable Area	Light gray, labeled Unavailable
Allocated Area	Stack elements with an allocation type of Allocated Area have different colors, based on the highlightField.
Available - Remaining Area	Bluish-green color (#CBE9EA). See Group Datasource configuration object for more information.

Note: Two allocation types - *Usable Area - Leased* and *Usable Area Owned* - do not appear in the stack diagram as stack elements, but are part of the floor and building statistics.