DataLayer

1. ArrayStore

- The ArrayStore is a store that provides an interface for loading and editing an in-memory array and handling related events.
- Options :

  data : Specifies the store's associated array.

  errorHandler : Specifies the function that is executed when the store throws an error.

  Key : Specifies the key property (or properties) that provide(s) key values to access data items. Each key value must be unique.

  Type: String | Array<String>

- Methods :

  byKey(key) : Gets a data item with a specific key.

  Parameters : key (Object | String | Number)

  Return Value : Promise (jQuery or native)

  clear() : Clears all the ArrayStore's associated data.

  createQuery() : Creates a Query for the underlying array.

  Return Value: Object (The Query object)

  insert(values) : Adds a data item to the store.

  Parameters : Values (Object) A data item.

  Return Value : Promise (jQuery or native)

  key() : Gets the key property (or properties) as specified in the key property.

  Return Value: any (The key property's value.)

  keyOf(obj) : Gets a data item's key value.

  Parameters : obj (Object) A data item.

  Return Value : any (The data item's key value.)

  load() : Starts loading data.

  Return Value : Promise (jQuery or native)

  load(options) : Starts loading data.

  Parameters : options (LoadOptions) Data processing settings.

  Return Value : Promise (jQuery or native)

  push(changes) : Pushes data changes to the store and notifies the DataSource.

  Parameters : changes (Array) Data changes to be pushed.

Each data change is an object that can have the following fields:

type : String (Data change type: "insert", "update", or "remove".)

data : Object (Changes that should be applied to the store's data.)

key : any (The key of the data item being updated or removed.)

index : Number (The position at which to display a new data item in a UI component bound to the store.)

remove(key) : Removes a data item with a specific key from the store.

Parameters : key (Object | String | Number) A data item's key value.

Return Value : Promise<void> (jQuery or native)

totalCount(options) : Gets the total count of items the load() function returns.

Parameters : obj (Object) Filtering and grouping properties.

Return Value : Promise<Number> (jQuery or native)

update(key, values) : Updates a data item with a specific key.

Parameters : key (Object | String | Number) A data item's key value.

values (Object) New values for the data item.

Return Value : Promise<any> (jQuery or native)

- Events :

onInserted : A function that is executed after a data item is added to the store.

Parameters : values(Object) The added data item.

Key (Object | String | Number) The item's key.

onInserting : A function that is executed before a data item is added to the store.

Parameters : values(Object) The added data item.

onload : A function that is executed after data is loaded to the store.

Parameters : result(Array) The loaded data.

onloading : A function that is executed before data is loaded to the store.

Parameters : loadOptions(LoadOptions) Data processing settings.

loadOptions (filter, sort, select, skip, take)

onModified : A function that is executed after a data item is added, updated, or removed from the store.

onModifying : A function that is executed before a data item is added, updated, or removed from the store.

onPush : The function executed before changes are pushed to the store.

Parameters : changes(Array)

onRemoved : A function that is executed after a data item is removed from the store.

Parameters : key(Object | String | Number) The removed data item's key.

onRemoving : A function that is executed before a data item is removed from the store.

Parameters : key(Object | String | Number) The key of the data item to be removed.

onUpdated : A function that is executed after a data item is updated in the store.

Parameters : key(Object | String | Number) The updated data item's key.

Values(Object) Updated values.

onUpdating : A function that is executed before a data item is updated in the store.

Parameters : key(Object | String | Number) key of data item to updated.

values (Object) New values for the data item fields.

2. CustomStore

- The CustomStore enables you to implement custom data access logic for consuming data from any source.
- Options :

byKey : Specifies a custom implementation of the byKey(key) method.

Parameters : key(Object | String | Number) A key value.

Return Value: Promise (jQuery or native).

cacheRawData : Specifies whether raw data should be saved in the cache. Applies only if loadMode is "raw".

Default Value : true

Data caching allows the CustomStore to decrease the number of data requests. On the downside, cached data and data in your source may become out of sync. If keeping them synchronized is crucial in your scenario, disable data caching by setting the cacheRawData property to false. In this case, the CustomStore will send a request for data on every call of the load, byKey and totalCount functions.

errorHandler : Specifies the function that is executed when the store throws an error.

This function accepts a JavaScript Error object as the parameter.

Insert : Specifies a custom implementation of the insert(values) method.

Parameters : values (Object) The data item to be inserted.

Return Value : Promise (jQuery or native).

key : Specifies the key property (or properties) that provide(s) key values to access data items. Each key value must be unique.

Type: String | Array<String>

load : Specifies a custom implementation of the load(options) method.

Parameters : options (LoadOptions) Data processing settings.

Return Value : Promise (jQuery or native) | Array.

loadMode : Specifies how data returned by the load function is treated.

Default Value : 'processed'

Accepted Values : 'processed' | 'raw'

Specify this property depending on the behavior you implemented for the load function. If this function sends data shaping properties to the server and fetches processed data, then loadMode should be "processed". If the load function simply fetches raw, unprocessed data from the server, set loadMode to "raw". In this case, the raw data will be processed on the client automatically.

remove : Specifies a custom implementation of the remove(key) method.

Parameters : key (Object | String | Number) key of data item to removed.

Return Value : Promise (jQuery or native).

totalCount : Specifies a custom implementation of totalCount(options) method.

Parameters : loadOptions (Object) Filtering and grouping settings.

update : Specifies a custom implementation of the update(key, values) method.

Parameters : key (Object | String | Number) key of data item to updated.

values (Object) An object with new values for data item.

Return Value : Promise (jQuery or native).

useDefaultSearch : Specifies whether the store combines the search and filter expressions. Defaults to true if the loadMode is "raw" and false if it is "processed".

Default Value : undefined

- Methods :

byKey(key) : Gets a data item with a specific key.

Parameters : key(Object | String | Number) A data item's key value.

Return Value : Promise (jQuery or native).

clearRawDataCache() : Deletes data from the cache.

Takes effect only if the cacheRawData property is true.

insert(values) : Adds a data item to the store.

Parameters : values(Object) A data item.

Return Value : Promise (jQuery or native).

key() : Gets the key property (or properties) as specified in the key property.

Return Value : any (The key property's value).

keyOf(obj) : Gets a data item's key value.

Parameters : obj(Object) A data item.

Return Value : any (The data item's key value).

load() : Starts loading data.

Return Value : Promise (jQuery or native).

load(options) : Starts loading data.

Parameters : options(LoadOptions) Data processing settings.

Return Value : Promise (jQuery or native).

push(changes) : Pushes data changes to the store and notifies the DataSource.

Parameters : changes(Array) Data changes to be pushed.

remove(key) : Removes a data item with a specific key from the store.

Parameters : key(Object | String | Number) A data item's key value.

Return Value : Promise<void> (jQuery or native).

totalCount(options) : Gets the total count of items the load() function returns.

Parameters : obj(Object) Filtering and grouping properties.

update(key, values) : Updates a data item with a specific key.

Parameters : key(Object | String | Number) A data item's key value.

values : Object (An object with new values for the data item).

Return Value : Promise (jQuery or native).

- Events :

onInserted : A function that is executed after a data item is added to the store.

Parameters : values(Object) The added data item.

Key (Object | String | Number) The item's key.

onInserting : A function that is executed before a data item is added to the store.

Parameters : values(Object) The added data item.

onload : A function that is executed after data is loaded to the store.

Parameters : result(Array) The loaded data.

onloading : A function that is executed before data is loaded to the store.

Parameters : loadOptions(LoadOptions) Data processing settings.

loadOptions (filter, sort, select, skip, take)

onModified : A function that is executed after a data item is added, updated, or removed from the store.

onModifying : A function that is executed before a data item is added, updated, or removed from the store.

onPush : The function executed before changes are pushed to the store.

Parameters : changes(Array)

onRemoved : A function that is executed after a data item is removed from the store.

Parameters : key(Object | String | Number) The removed data item's key.

onRemoving : A function that is executed before a data item is removed from the store.

Parameters : key(Object | String | Number) The key of the data item to be removed.

onUpdated : A function that is executed after a data item is updated in the store.

Parameters : key(Object | String | Number) The updated data item's key.

values(Object) Updated values.

onUpdating : A function that is executed before a data item is updated in the store.

Parameters : key(Object | String | Number) key of data item to updated.

values (Object) New values for the data item fields.

- LoadOptions :

customQueryParams : An object for storing additional settings that should be sent to the server. Relevant to the ODataStore only.

expand : An array of strings that represent the names of navigation properties to be loaded simultaneously with the ODataStore.

filter : A filter expression. Defines filtering parameters. Possible variants:

Binary filter : Supported operators: "=", "<>", ">", ">=", "<", "<=", "startswith", "endswith", "contains", ”notcontains".

Unary filter : Supported operators: binary operators, "!".

Complex filter : Supported operators: binary and unary operators, "and", "or".)

group : A group expression. Defines grouping levels to be applied to the data.

This object can have the following fields:

selector : String (The field name to group by.

desc : Boolean (Defines the selector field's descending sort order).

isExpanded : Boolean Defines whether to return the group's items or null. For grid-like components (DataGrid, TreeList, and so on), this field always returns true for all groups except the last one. DevExtreme components accepts the unspecified value of the isExpanded field as true.

groupInterval : Number or String

A numeric value groups data in ranges of the given length. A string value applies only to dates and can be "year", "quarter", "month", "day", "dayOfWeek", "hour", "minute" and "second".

groupSummary : A group summary expression. Used with the group setting.

Contains group summary definitions with the following structure, where summaryType can be "sum", "avg", "min", "max" or "count":

When this property is specified, each data object should have a summary array that contains the resulting values in the same order as the summary definitions.

parentIds : The IDs of the rows being expanded. Relevant only when the CustomStore is used in the TreeList UI component.

Type : Array

If the TreeList's expandedRowKeys are set, the parentIds array contains them and the root ID. Otherwise, the array contains only the ID of the row being expanded.

requireGroupCount : Indicates whether a top-level group count is required. Used in conjunction with the filter, take, skip, requireTotalCount, and group settings.

Type : Boolean

When this property is true, the result should contain the groupCount field. This field is the resulting data set's top-level group count.

requireTotalCount : Indicates whether the total count of data objects is needed.

Type : Boolean

When this property is true, the store expects the result to contain the totalCount field, which is the total data object count in the resulting data set. This count should reflect the number of data items after filtering but disregard any take parameter used for the query.

searchExpr : A data field or expression whose value is compared to the search value.

Type : getter | Array<getter>

searchOperation : A comparison operation. Can have one of the following values: "=", "<>", ">", ">=", "<", "<=", "startswith", "endswith", "contains", "notcontains", "isblank" and "isnotblank".

Type : String

searchValue : The current search value.

select : A select expression.

skip : The number of data objects to be skipped from the result set's start. In conjunction with take, used to implement paging.

Type : Number

sort : A sort expression.

selector **:** String (The field name to sort by).

desc : Boolean (Defines the selector field's descending sort order).

take : The number of data objects to be loaded. In conjunction with skip, used to implement paging.

Type: Number

totalSummary : A total summary expression.

Contains summary definitions with the following structure, where summaryType can be "sum", "avg", "min", "max" or "count":

When this property is specified, the store expects the result to have the summary array that contains the result values in the same order as the summary definitions.

userData : An object for storing additional settings that should be sent to the server.

3. DataSource

- The **DataSource** is an object that provides an API for processing data from an underlying store.
- Options :

  customQueryParams : An object for storing additional settings that should be sent to the server. Relevant to the ODataStore only.

  expand : An array of strings that represent the names of navigation properties to be loaded simultaneously with the ODataStore.

  filter : A filter expression. Defines filtering parameters.

  group : A group expression. Defines grouping levels to be applied to the data.

  This object can have the following fields:

  selector : String (The field name to group by).

  desc : Boolean (Defines the selector field's descending sort order).

  map : Specifies an item mapping function.

  Parameters : dataItem(Object) An initial data item.

  Return Value : Object (A modified data item).

  pageSize : Specifies the maximum number of data items per page. Applies only if paginate is true.

  Default Value: 20

  When data is grouped, this property specifies the number of groups per page.

  paginate : Specifies whether the DataSource loads data items by pages or all at once. Defaults to false if group is set; otherwise, true.

  postProcess : Specifies a post processing function.

  Parameters : data(Array) Data loaded in the DataSource.

  Return Value : Array (Data after processing).

pushAggregationTimeout : Specifies the period (in milliseconds) when changes are aggregated before pushing them to the DataSource.

When this property is undefined, the aggregation period is calculated automatically based on the rendering speed's measurements.

requireTotalCount : Specifies whether the DataSource requests the total count of data items in the storage.

reshapeOnPush : Specifies whether to reapply sorting, filtering, grouping, and other data processing operations after receiving a push.

Default Value: false

searchExpr : A data field or expression whose value is compared to the search value.

Type : getter | Array<getter>

searchOperation : A comparison operation. Can have one of the following values: "=", "<>", ">", ">=", "<", "<=", "startswith", "endswith", "contains", "notcontains", "isblank" and "isnotblank".

searchValue : The current search value.

select : A select expression.

sort : A sort expression.

selector : String (The field name to sort by).

desc : Boolean (Defines the selector field's descending sort order).

store : Configures the store underlying the DataSource.

Type : Store | Store Configuration | Array | any

Store instance

An ArrayStore, LocalStore, ODataStore, or CustomStore instance.

Store configuration object

An ArrayStore, LocalStore, or ODataStore configuration object.

Make sure to set the type property.

Array

Assigning an array to the store property automatically creates an ArrayStore in the DataSource.

- Methods :

cancel(operationId) : Cancels the load operation with a specific identifier.

Return Value : Boolean

true if the operation was canceled; false if it was not found.

dispose() : Disposes of all the resources allocated to the DataSource instance.

filter()/(filterExpr) **:** Gets the filter property's value if parametes are null otherwise sets parameters as filterExpr.

Pass null to clear filtering settings.

group()/(groupExpr) : Gets the group property's value if parametes are null otherwise sets parameters as groupExpr.

isLastPage() : Checks whether the count of items on the current page is less than the pageSize. Takes effect only with enabled paging.

Return Value : Boolean

true if the item count is less than the pageSize; otherwise false.

isLoaded() : Checks whether data is loaded in the DataSource.

Return Value : Boolean

true if data is loaded; otherwise false.

isLoading() : Checks whether data is being loaded in the DataSource.

Return Value : Boolean

true if data is being loaded; otherwise false.

items() : Gets an array of data items on the current page.

Return Value : Array (Data items).

key() : Gets the value of the underlying store's key property.

Return Value : Object | String | Number (A key expression).

load() : Starts loading data.

Return Value : Promise (jQuery or native).

loadOptions() : Gets an object with current data processing settings.

Return Value : Object

pageIndex()/(newIndex) : Gets/Sets the current page index.

A zero-based page index.

pageSize()/(value) : Gets/Sets the page size.

paginate()/(value) : Gets/Sets the paginate property's value.

reload() : Clears currently loaded DataSource items and calls the load() method.

Return Value : Promise (jQuery or native).

requireTotalCount()/(value) : Gets/Sets the requireTotalCount property's value.

searchExpr()/searchExpr(expr) : Gets/Sets the searchExpr property's value.

searchOperation()/(op) : Gets/Sets the searchOperation property's value.

searchValue()/(value) : Gets/Sets the searchValue property's value.

select()/(expr) : Gets/Sets the select property's value.

sort()/(sortExpr) : Gets/Sets the sort property's value.

store() : Gets the instance of the store underlying the DataSource.

Return Value : Object (A store instance).

totalCount() : Gets the number of data items in the store after the last load() operation without paging. Takes effect only if requireTotalCount is true.

Return Value : Number (The number of data items).

- Events :

onChanged : A function that is executed after data is loaded.

Parameters : e(Object) Information about changes.

Appears only when the push(changes) method is called and the reshapeOnPush property is false.

onLoadError : A function that is executed when data loading fails.

Parameters : error(Object) The error.

onLoadingChanged : A function that is executed when the data loading status changes.

Parameters : isLoading(Boolean) Indicates whether data is being loaded.

4. LocalStore

- The **LocalStore** is a store that provides an interface for loading and editing data from HTML Web Storage (also known as window.localStorage) and handling related events.
- Options :

data : Specifies the store's associated array.

Type : Array

errorHandler : Specifies the function that is executed when the store throws an error.

Type : Function

flushInterval : Specifies a delay in milliseconds between when data changes and the moment these changes are saved in the local storage. Applies only if immediate is false.

Type : Number

Default Value : 10000

Immediate : Specifies whether the LocalStore saves changes in the local storage immediately.

Type : Boolean

Default Value : false

Key : Specifies the key property (or properties) that provide(s) key values to access data items. Each key value must be unique.

Type : String | Array<String>

Name : Specifies the name under which data should be saved in the local storage. The dx-data-localStore- prefix will be added to the name.

Type : String

- Methods :

Refer to ArrayStore methods.

- Events :

Refer to ArrayStore events.

5. Query

- The Query is an object that provides a chainable interface for making data queries.
- To create a Query, call the query(array) or query(url, queryOptions) method, depending on the type of the storage you access. The Query supports method chaining. This enables you to execute several methods in a single statement.
- Methods :

aggregate(seed, step, finalize) : Calculates a custom summary for all data items.

Parameters : seed(Object) The initial value.

step : Function (A function called for each item).

finalize : Function (Called after the calculation is finished).

Return Value : Promise (jQuery or native).

aggregate(step) : Calculates a custom summary for all data items.

Parameters : step(Function) A function that is called for each item.

Return Value : Promise (jQuery or native).

This is a shortcut for the aggregate(seed, step, finalize) method. It omits the seed and finalize parameters: instead of the seed value, the accumulator value is initialized with the first item's value; the finalize parameter's omission means that the calculation result is the accumulator value after the last step function's execution.

avg()/(getter) : Calculates the average of all values or of all values found using a getter.

Return Value : Promise<Number> (jQuery or native).

count() : Calculates the number of data items.

Return Value : Promise<Number> (jQuery or native).

enumerate() : Executes the Query. This is an asynchronous alternative to toArray() method. Applies only to numeric arrays.

Return Value : Promise (jQuery or native).

filter(criteria) : Filters data items using a filter expression.

Parameters : criteria(Array) A filter expression; described in the Filtering section.

Return Value : Query

filter(predicate) : Filters data items using a custom function.

Parameters : predicate(Function) A function that accepts a data item and returns true if it should be included in the resulting array and false otherwise.

Return Value : Query

groupBy(getter) : Groups data items by the specified getter.

Parameters : getter(Object) A getter; in most cases, the name of data field to group by.

Return Value : Query

max()/(getter) : Calculates the maximum of all values or found using a getter.

Applies only to numeric arrays.

Return Value : Promise<Number | Date> (jQuery or native).

min()/(getter) : Calculates the minimum of all values or found using a getter.

Applies only to numeric arrays.

Return Value : Promise<Number | Date> (jQuery or native).

select(getter) : Selects individual fields from data objects.

Parameters : getter(Object) A getter; in most cases, the names of the data fields to select.

Return Value : Query

slice(skip, take) : Gets a specified number of data items starting from a given index.

Parameters : skip(Number) The index of the first data item to get.

take(Number | undefined) Optional. The number of data items to get.

Return Value : Query

sortBy(getter) : Sorts data items by the specified getter in ascending order.

Parameters : getter(Object) A getter; in most cases, the name of the data field to sort by.

Return Value : Query

sortBy(getter, desc) : Sorts data items by the specified getter in the specified sorting order.

Parameters : getter(Object) A getter; in most cases, the name of the data field to sort by.

desc(Boolean) Pass **true** to sort in descending order.

Return Value : Query

sum()/(getter) : Calculates the sum of all values or found using a getter.

Applies only to numeric arrays.

Return Value : Promise<Number> (jQuery or native)

thenBy(getter) : Sorts data items by one more getter in ascending order.

Parameters : getter(Object) A getter; in most cases, the name of the data field to sort by.

Return Value : Query

thenBy(getter, desc) : Sorts data items by one more getter in the specified sorting order.

Parameters : getter(Object) A getter; in most cases, the name of the data field to sort by.
desc(Boolean) Pass true to sort in descending order.

Return Value : Query

toArray() : Gets data items associated with the Query. This is a synchronous alternative to the enumerate() method.

Return Value : Array