Using PCore and PConnect Public APIs '24.2

5 September 2025

CONTENTS

Caching and invalidating data pages in Constellation

Optimize data page usage and leverage browser memory to enhance an application’s performance.

The getDataAsync and getPageDataAsync APIs are used to retrieve data from data pages using the `/data_views` DX API end point. These APIs cache the fetched data in the browser’s memory.

Client-side caching takes place only when both the following scenarios are met:

- The **Reload once per interaction** checkbox is cleared for the data page.
- The `query` object within the getDataAsync and getPageDataAsync APIs contains only one key.

  For example, in the following code, the `query` object contains only the `select` key.

```
const dataViewName = "D_EmployeeList";
const parameters = {
  "dept": "Engineering"
};
const paging = {
  "pageNumber": 1,
  "pageSize": 10
};
const query = {
  "select": [{
      "field": "Name"
    },
    {
      "field": "Role"
    }
  ]
};
const context = getPconnect().getContext();
PCore.getDataPageUtils().getDataAsync(dataViewName, context, parameters, p
aging, query);
```

When subsequent calls are made with the same query and parameters, the results are retrieved directly from the cache resulting in an improved response time.

- **Invalidating cached data pages**
- **Cache invalidation scenarios**

Related reference

- getDataAsync(dataPageName, context, parameters, paging, query, options)
- getPageDataAsync(dataPageName, context, parameters, options)

Invalidating cached data pages

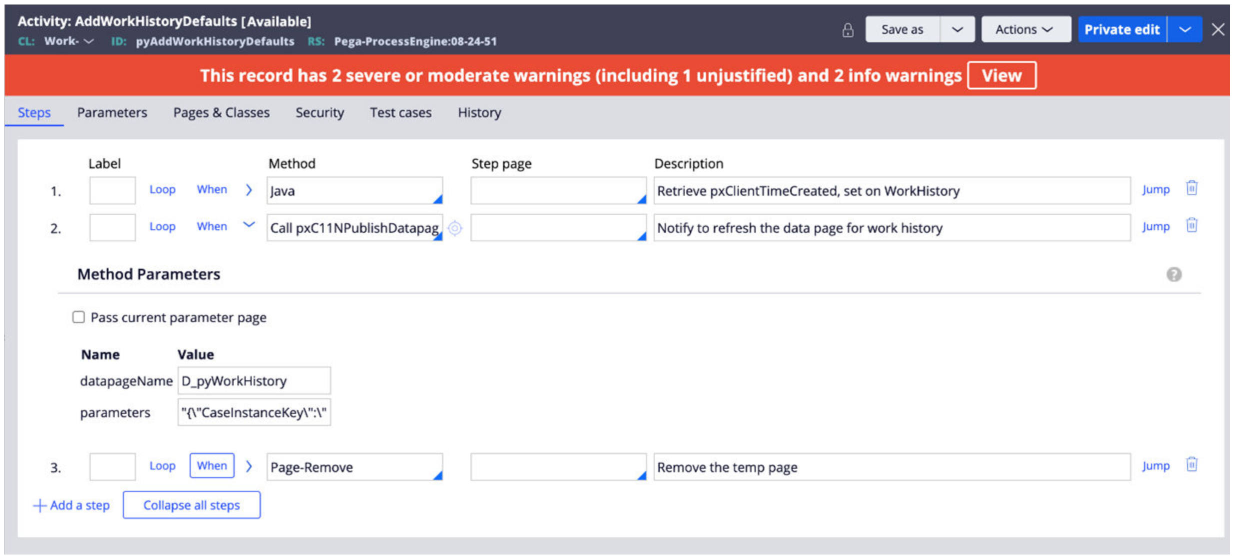
Learn how the client-side cached data is invalidated when a data page is updated.

When you update a data page, call the `pxC11NPublishDatapageUpdate` activity. This activity publishes the `DATAPAGE_UPDATED` web socket message. This web socket message invalidates the cached data.

The `pxC11NPublishDatapageUpdate` activity contains two parameters:

- `datapageName` - The name of the data page in string format.
- `parameters` - The stringified JSON object of the parameters for the data page.

For example, for a `D_pyWorkHistory` data page that accepts the `CaseInstanceKey` parameter, the stringified JSON object is `"{\"CaseInstanceKey\":\"" +.pzInsKey+ "\"}"`.



Activity: AddWorkHistoryDefaults [Available]
CL: Work- ID: pyAddWorkHistoryDefaults RS: Pega-ProcessEngine:08-24-51

This record has 2 severe or moderate warnings (including 1 unjustified) and 2 info warnings

	Label			Method	Step page	Description
1.		Loop	When	Java		Retrieve pxClientTimeCreated, set on WorkHistory
2.		Loop	When	Call pxC11NPublishDatapag		Notify to refresh the data page for work history

Method Parameters

Pass current parameter page

Name	Value
datapageName	D_pyWorkHistory
parameters	"{\"CaseInstanceKey\":\"

	Label			Method	Step page	Description
3.		Loop	When	Page-Remove		Remove the temp page

Stringified parameters

For example, if a data page has multiple parameters, the stringified JSON object is formed using the following syntax: `"{\"Department\":\"" +.Department+ "\",\"Role\":\"" +.Role+ "\"}"`

NOTE: You can subscribe or unsubscribe to changes to a data page by using the following APIs:

- subscribeToDataPageUpdates(subscriptionId, callback, dataPageName, parameters)
- unsubscribeToDataPageUpdates(subscriptionId, dataPageName, parameters)

Related reference

- subscribeToDataPageUpdates(subscriptionId, callback, dataPageName, parameters)
- unsubscribeToDataPageUpdates(subscriptionId, dataPageName, parameters)

Cache invalidation scenarios

Learn how the behavior of client-side cache invalidation varies based on the parameters passed to the activity.

When a data page is updated, the `pxC11NPublishDatapageUpdate` activity must be invoked to invalidate the cache. The cache is invalidated based on the parameters passed to the activity. Let us go through several scenarios to examine how the behavior of cache invalidation varies.

Consider a list type data page `D_EmployeeList` that is used to fetch data using the getDataAsync API. The `D_EmployeeList` data page contains a `department` parameter.

In each of the following scenarios, a different value is passed to the `department` parameter and the data fetched is cached.

Scenario 1

The value of `department` is `Engineering`.

1. The getDataAsync API is called with `parameters`.
```
const dataPageName = "D_EmployeeList";
const parameters = {
  "department": "Engineering"
};
const context = getPConnect().getContext();
PCore.getDataPageUtils().getDataAsync(dataPageName, context, parameters);
```
2. The getDataAsync API is called with `parameters` and `paging`.

```
const dataPageName = "D_EmployeeList";
const parameters = {
  "department": "Engineering"
};
const paging = {
  "pageNumber": 1,
  "pageSize": 10
};
const context = getPConnect().getContext();
PCore.getDataPageUtils().getDataAsync(dataPageName, context, parameters, p
aging);
```

3. The getDataAsync API is called with `parameters`, `paging`, and `query`.

```
const dataPageName = "D_EmployeeList";
const parameters = {
  "department": "Engineering"
};
const paging = {
  "pageNumber": 1,
  "pageSize": 10
};
const query = {
  "select": [{
      "field": "Name"
    },
    {
      "field": "Role"
    },
    {
      "field": "Gender"
    }
  ]
};
const context = getPConnect().getContext();
PCore.getDataPageUtils().getDataAsync(dataPageName, context, parameters, p
aging, query);
```

Scenario 2

The value of `department` is `HR`.

1. The getDataAsync API is called with `parameters`.

```
const dataPageName = "D_EmployeeList";
const parameters = {
  "department": "HR"
};
const context = getPConnect().getContext();
PCore.getDataPageUtils().getDataAsync(dataPageName, context, parameters);
```

2. The getDataAsync API is called with `parameters` and `paging`.

```
const dataPageName = "D_EmployeeList";
const parameters = {
  "department": "HR"
};
const paging = {
  "pageNumber": 1,
  "pageSize": 10
};
const context = getPConnect().getContext();
PCore.getDataPageUtils().getDataAsync(dataPageName, context, parameters, p
aging);
```

3. The getDataAsync API is called with `parameters`, `paging`, and `query`.

```
const dataPageName = "D_EmployeeList";
const parameters = {
  "department": "HR"
};
const paging = {
  "pageNumber": 1,
  "pageSize": 10
};
const query = {
  "select": [{
        "field": "Name"
    },
    {
      "field": "Role"
    },
    {
      "field": "Gender"
    }
  ]
};
const context = getPConnect().getContext();
PCore.getDataPageUtils().getDataAsync(dataPageName, context, parameters, p
aging, query);
```

Scenario 3

An empty object is passed as `parameters`.

1. The getDataAsync API is called with empty `parameters`.

```
const dataPageName = "D_EmployeeList";
const parameters = {};
const context = getPConnect().getContext();
PCore.getDataPageUtils().getDataAsync(dataPageName, context, parameters);
```

2. The getDataAsync API is called with empty `parameters` and `paging`.

```
const dataPageName = "D_EmployeeList";
const parameters = {};
const paging = {
  "pageNumber": 1,
  "pageSize": 10
};
const context = getPConnect().getContext();
PCore.getDataPageUtils().getDataAsync(dataPageName, context, parameters, p
aging);
```

3. The getDataAsync API is called with empty `parameters`, `paging`, and `query`.

```
const dataPageName = "D_EmployeeList";
const parameters = {};
const paging = {
  "pageNumber": 1,
  "pageSize": 10
};
const query = {
  "select": [{
      "field": "Name"
    },
    {
      "field": "Role"
    },
    {
      "field": "Gender"
    }
  ]
};
const context = getPConnect().getContext();
PCore.getDataPageUtils().getDataAsync(dataPageName, context, parameters, p
aging, query);
```

Cache invalidation behavior

Let us consider the above scenarios where the data fetched from all the sub-scenarios was cached.

- When the `pxC11NPublishDatapageUpdate` activity is executed by passing `D_EmployeeList` as the `datapageName` and `"{\"department\":\"Engineering\"}"` as the `parameters`, the data cached from all the sub-scenarios of Scenario 1 and Scenario 3 are invalidated.
- When the `pxC11NPublishDatapageUpdate` activity is executed by passing `D_EmployeeList` as the `datapageName` and `"{\"department\":\"HR\"}"` as the `parameters`, the data cached from all the sub-scenarios of Scenario 2 and Scenario 3 are invalidated.
- When the `pxC11NPublishDatapageUpdate` activity is executed by only passing `D_EmployeeList` as the `datapageName`, the data cached from all the sub-scenarios of Scenario 1, Scenario 2, and Scenario 3 are invalidated.

NOTE: For page type data pages, caching can be performed through the getPageDataAsync API and the cache invalidation behavior is the same as for list type data pages.

Related reference

- getDataAsync(dataPageName, context, parameters, paging, query, options)
- getPageDataAsync(dataPageName, context, parameters, options)