Using PCore and PConnect Public APIs '24.2

5 September 2025

CONTENTS

Working with Containers **3**
Container Data Model. 3
Life cycle of a container. 7
Retrieving information from containers. 14

Working with Containers

Use containers to present dynamic content on the screen. Learn about the structure of a container, the various stages of its life cycle, and how to retrieve information from it.

A container is a component that comprises various items. Based on the type of container, it can contain single or multiple items. Each container item has a data context and unique identifier.

The data context of a container item can be shared with a child container item or a related container item as needed. Each container maintains the order in which its items were accessed.

If a container contains multiple items, it displays the only active item. If we delete the active item from the container, it displays the most recently used item that becomes active in the container.

- **Container Data Model**
- **Life cycle of a container**
- **Retrieving information from containers**

Container Data Model

View the structure of a container in the container data model. This structure comprises the type of the container, its access order, and its items.

Data

This contains the data context associated with container items opened in the application.

Containers

This contains information about the usage and contents of each container used in the application.

Each container contains the following entries:

- Type
- accessedOrder
- Items

Type

This represents the type of the container.

There are two types of containers:

- Single containers that contain only one item.
- Multiple containers that contain multiple items.

accessedOrder

This is an array that contains the fully qualified names of container items in the order they were accessed. This order is used to activate the most recently used container item when the current active item is deleted.

> **NOTE:** The accessedOrder array contains the container item names in the order where the first entry in the array is the first item accessed and the last entry in the array is the most recently accessed item.

Items

This is a map containing the information about the specific items in a container. Each entry in the map contains the root view information of the corresponding item, a

semantic URL which contains the unique ID of the item, and the data context from the primary data entry of the Store for the corresponding item.

Usage example

In this example, there are two containers, `app/primary` and `app/primary_2/workarea`. The `app/primary` container has been used to access the `app/primary_1` and `app/primary_2` items. Since the `app/primary_2` item was most recently accessed, it appears last in the `accessedOrder` array.

You can view the contents of the `app/primary_2` item from its entry in the `app/primary` item structure. The `app/primary_2` item is associated with a `view` containing the `pyDetails` information about the content of a `caseInfo` object. This is represented with a semantic URL `RequestApproval/ RA-123`.

The information about the `caseInfo` object can be found in the information of the `app/primary_2` item found in the `data` key of the Store.

```
{
  data: {
    app: {

    },
    'app/primary_1': {
      ...
    },
    'app/primary_2': {
      pxRequestor: {
        pxUserName: 'abc',
        pxUserIdentifier: 'abc@pegasystems.com'
      },
      caseInfo: {
        content: {
```

```
        ...
      }
    }
  }
},
containers: {
  'app/primary': {
    type: 'multiple',
    accessedOrder: [
      'app/primary_1',
      'app/primary_2'
    ],
    items: {
      'app/primary_1': {
        ...
      },
      'app/primary_2': {
        view: {
          type: 'reference',
          config: {
            name: 'pyDetails',
            context: 'caseInfo.content',
            type: 'view'
          }
        },
        semanticURL: 'RequestApproval/{workID}',
        key:'OE7UD6-SPACETRAVEL-WORK RA-123',
        context: 'app/primary_2'
      }
    }
  },
  'app/primary_2/workarea':{
```

```
    ...
   }
  }
}
```

Life cycle of a container

Use a container to perform various actions. This section describes the stages in the life cycle of a container as a series of steps.

The five stages in the life cycle of a container are:

- Initialize a container
- Add items to a container
- Update items in a container
- Activate items in a container
- Delete items from a container

Initialize a container

To utilize a container, you must initialize it. You can initialize a container in a Store using the initializeContainers API in the ContainerManager class. For more information, see initializeContainers(containerInfo).

After initialization, the containers entry will have the following structure:

```
{
  containers: {
    'app/primary': {
      type: 'multiple',
      accessedOrder: [],
      items: {}
    }
```

```
    }
}
```

Add items to a container

You can add items to an initialized container to populate it with data. You can add items to a container using the addContainerItem API in the ContainerManager class. For more information, see addContainerItem(containerInfo).

The payload passed through the addContainerItem API contains the context that can be obtained through the getContextName API and the key that is the unique ID for the container item.

Consider an item added to a container:

```
const containerManager = getPConnect().getContainerManager();
containerManager.addContainerItem({
  context: 'app/primary_1',
  key: 'OE7UD6-SPACETRAVEL-WORK RA-123',

  data: {
   caseInfo: {
     ...
   }
 }
});
```

After adding the item to the container, the containers entry will have the following structure:

```
{
 data: {
  app/primary_1: {
```

```
    caseInfo: {
      ...
    }
  }
 },
 containers: {
  'app/primary': {
    type: 'multiple',
    accessedOrder: [
     'app/primary_1'
    ],
    items: {
     'app/primary_1': {
       view: {
         type: 'reference',
         config: {
           name: 'pyHome',
           type: 'view',
           ruleClass: 'Data-Portal'
         }
       },
       key: ' OE7UD6-SPACETRAVEL-WORK RA-123',
       context: 'app/primary_1'
     }
    }
  }
 }
}
```

Update items in a container

You can update an item in a container using the updateContainerItem API in the ContainerManager class. For more information, see updateContainerItem(containerInfo).

The payload passed through the updateContainerItem API contains the context that can be obtained through the getContextName API and the key that is the unique ID for the container item.

Usage example

In this example, the `containerInfo` object contains the data to update the `workarea_0` item in the `app/primary_0/workarea` container.

```
getPConnect().getContainerManager().updateContainerItem(containerInfo);
```

The payload in the `containerInfo` object contains the target container that contains the item to be updated and the `containerItemID` of the item to be updated.

The APIs in the ContainerUtils class can be used to prepare the argument for the updateContainerItem API. The data can be obtained using Constellation DX APIs or external sources based on the item to be updated.

The `containerInfo` object has the following structure:

```
{
   target:"app/primary_0/workarea",
   containerItemID:"app/primary_0/workarea_0",
   context:"app/primary_0",
   data: {...}
}
```

The state before the updateContainerItem API is called is as shown below:

```
{
  data: {
    app/primary_2: {
      content: {
        FirstName: "",
        Lastname: "",
        Email: ""
      }
    }
  },
  containers: {
    'app/primary': {
      type: 'multiple',
      accessedOrder: [
        'app/primary_1'
      ],
      items: {
        'app/primary_2': {
          view: {
            type: 'reference',
            config: {
              name: 'pyApplication',
              type: 'view',
              ruleClass: 'Data-Portal'
            }
          },
          key: 'OE7UD6-SPACETRAVEL-WORK RA-12',
          context: 'app/primary_2'
        }
      }
    }
```

```
  }
}
```

The `containerInfo` object is passed as shown below:

```
{
   target: 'app/primary_2/workarea',
   containerItemID: 'app/primary_2/workarea_1'',
   context: 'app/primary_2',
   data: {
     content: {
       FirstName: "James",
       Lastname: "Cameron",
       Email: "a.b@c.com"
     }
   }
}
```

The state after the updateContainerItem API is called is as shown below:

```
{
  data: {
    app/primary_2: {
      content: {
        FirstName: "James",
        Lastname: "Cameron",
        Email: "a@b.c"
      }
    }
  },
  containers: {
    'app/primary': {
```

```
    type: 'multiple',
    accessedOrder: [
     'app/primary_1'
    ],
    items: {
     'app/primary_2': {
       view: {
         type: 'reference',
         config: {
           name: 'pyApplication',
           type: 'view',
           ruleClass: 'Data-Portal'
         }
       },
       key: 'OE7UD6-SPACETRAVEL-WORK RA-12',
       context: 'app/primary_2'
     }
    }
   }
  }
}
```

Activate items in a container

When an item in a container is activated, the container displays the data or context associated with the target container and the `containerItemID`. You must activate an item before updating or deleting data in it.

Usage example

In this example, the `containerInfo` object contains the data to activate the `app/primary_1` item in the 'app/primary' container.

```
getPConnect().getContainerManager().activateContainerItem(containerInfo);
```

The `containerInfo` object has the following structure:

```
{
  target: 'app/primary',  // Target of the view container
  containerItemID: 'app/primary_1' // view container Context name to be activated
}
```

Delete items from a container

You can delete an item from a container using the removeContainerItem API in the ContainerManager class. For more information, see removeContainerItem(containerInfo).

When an item is deleted from a container, the item is removed from the `accessedOrder` array. If the deleted item was being displayed, the last entry in the `accessedOrder` array is now displayed.

Usage example

In this example, the `containerInfo` object contains the data to delete the `app/primary_2` item in the `app/primary` container.

```
getPConnect().getContainerManager().removeContainerItem(containerInfo);
```

The `containerInfo` object has the following structure:

```
{
  target: 'app/primary',
  containerItemID: 'app/primary_2'
}
```

Retrieving information from containers

Use the APIs in the ContainerUtils class to retrieve information pertaining to containers used in the application.

For more information, see APIs in the ContainerUtils class.