7 Planning

7.1 What to Expect from Analytics Designer Regarding Planning?

Analytics designer reuses the Planning features of SAP Analytics Cloud and leverage the capabilities by offering flexible scripting that supports customizations of applications according to user requirements. Planning Data Models, Allocations, Data Action Triggers, and all Planning features can be integrated to applications.

And what can you not expect? In analytics designer you can't use Input Tasks and Planning scripting isn't possible for models based on BPC Write-Back.

7.2 Basic Planning Concepts in Analytics Designer

Planning specific features can be triggered through the toolbar icons in the *Plan* area.

Figure 73: Toolbar Planning Features

These icons are greyed out if no table cell with a planning model is selected.

Most of these features can also be triggered through scripting.

To get the Planning Table object, use the script below. If the table has no planning model assigned, it will return `undefined`.

```
Table.getPlanning(): Planning | undefined
```

Scripting will perform the same planning actions that could be done via UI. The benefits of scripting are augmented in cases which you want to minimize the number of clicks from your user, personalize your UI or when a special customer requirement can't be fulfilled with standard planning behavior.

Data can't be changed during design time, and you can enable the usage of planning features during runtime in two different ways:

- In the table designer UI: You can find in the *Builder* panel, section *Properties*, a box called *Planning enabled*.

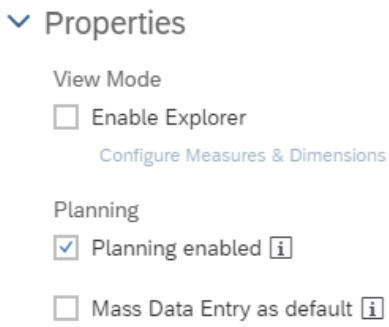


Figure 74: Planning Enabled

- Through the script below:

```
setEnabled(boolean): void
```

This option can be useful when you shall disable planning due to specific requirements. For example, budget might not be changed in last quarter of the year.

One other valuable script allows checking whether the data model is planning-enabled:

```
isEnabled(): boolean
```

In the Table's *Builder* panel, there are some configurations that you can do in each dimension, and **Unbooked** Data might be a good choice when, for example, your Planning Data Model has no booked data, and your end users need to see which dimension members are available for planning.

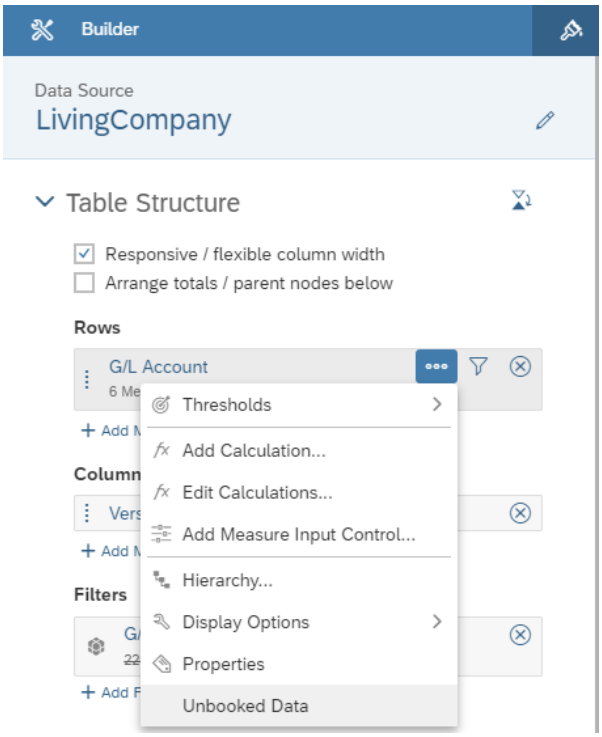


Figure 75: Unbooked Data

7.3 Refreshing Your Data

This feature isn't exclusive to Planning and affects all data models and widgets of your application. It can be reached in two different ways:

- By clicking on the first icon of the toolbar.

- Through the script below:

```
Application.refreshData(): void
```

Scripting is useful when, for example, you use data models with Live Connectivity and the end user wishes to refresh data because a background process that updates master data has finished after the application was opened.

7.4 Setting User Input

Instead of having to guide a business user by showing which table cell should be planned, the app designer could create a separate input field. This input field has a pre-informed value from a table selection. The changed value is then updated on a planning model.

The picture below represents the scenario mentioned above to explain this feature:

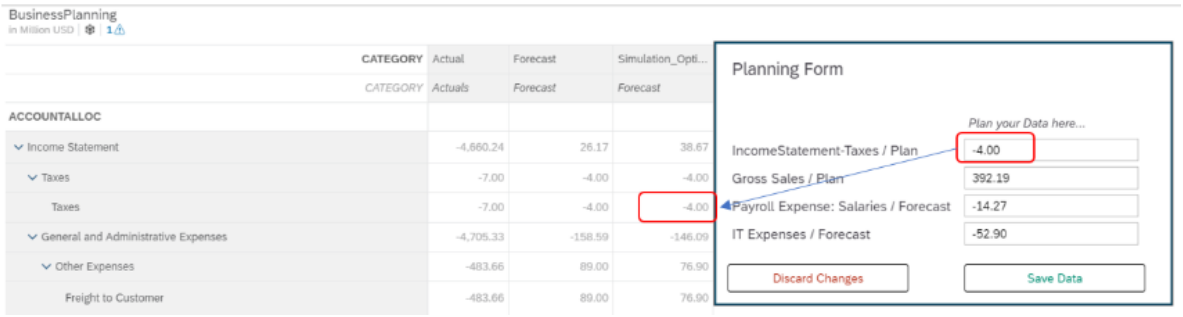


Figure 76: SetUserInput

In this example, the following scripting would be included on the *Save Data* button.

To update a cell of a table with the given value:

```
setUserInput(selectedData: Selection, value: String): boolean
```

Few considerations for this script:

- Value can be maximum 17 characters.
- If value is scaled, then it shall be less than 7 digits.
- It can be performed from a widget or from a table event.

Regarding data formatting – it takes as parameter either a raw value in the user formatting setting ("1234.567" with "." grouping separator) or a scale in the user formatting setting (for example, "*0.5" to divide the value by half or "*2" to double the value) – both of type string.

Example: (scaling in million)

- If you plan “12345678” the formattedValue will be “12.35”.
- If you plan “123456789” the formattedValue will be “123.46”.
- If you plan “*0.5” of rawValue “123456789” the rawValue will be “61728394.5” and formattedValue will be “61.73.

Regarding data validation:

- If an invalid value is planned, error/warning message is returned, and the script API also returns false.
- If the same value is planned twice, the value is set, and the script API returns true.
- If the cell is locked, the script API returns false, and a warning message is shown to the user.

To submit the updated cell data with all the beforehand modified data and to trigger refresh of data:

```
submitData(): boolean
```

7.5 Planning Versions

There are two types of planning versions, private and public.

7.5.1 Private Versions

This data isn’t visible to other users and other solutions of SAP Analytics Cloud.

```
getPrivateVersions(): [Array of Planning Private Versions] | empty array
getPrivateVersion(versionId: String): Planning Private Version | undefined
```

The script below returns the user ID of the user who created this private version.

```
getOwnerID(): String
```

7.5.2 Public Versions

This data is visible to all users and all solutions of SAP Analytics Cloud.

```
getPublicVersions(): [Array of Planning Public Versions] | empty array
getPublicVersion(versionId: String): Planning Public Version | undefined
```

Both planning version types have IDs.

```
getId(): String
```

You can use it, for example, when calling `getData()`.

```
getDisplayId(): String
```

You can use it, for example, to display the version in dropdowns or texts.

All versions but ‘Actual’ can be deleted.

```
deleteVersion(): boolean
```

7.6 How to Manage Versions

7.6.1 Publishing and Reverting Data Changes

Any change in data in any type of version is automatically saved. This means that even without any active saving action, if the browser is closed by mistake, for example, data will still be there when application is reopened by the same user who changed the data.

But to make this data visible to other users, you can publish the public versions through the following toolbar icon:

Figure 77: Publish Version

After clicking this icon, the dialog below is opened and an action can be taken per model. You can also revert, and all data changes will be discarded.

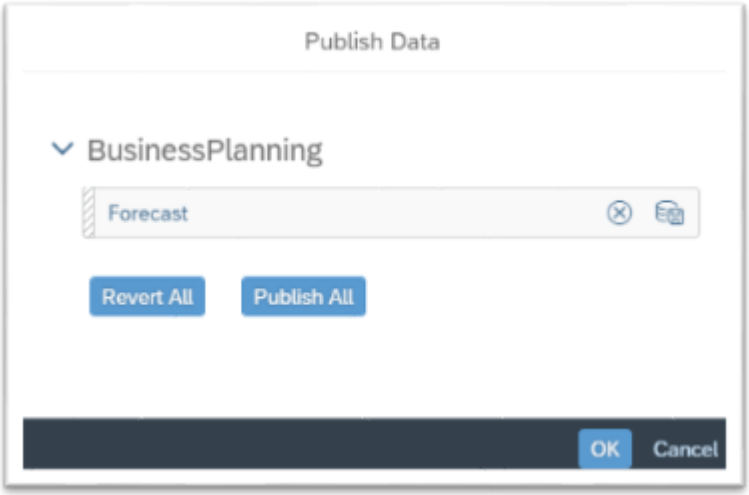


Figure 78: Publish Data

The actions performed within this dialog can also be done via the below scripting on public versions:

```
revert(): boolean
publish(): boolean
```

After the execution of these scripts, a message informs whether the script ended successfully or not. These are the expected messages:

Figure 79: Success Message

If the version wasn't modified before these actions are triggered, the message below should be expected:

Figure 80: Message

This message could be avoided if the dirty check is done in advance.

```
isDirty(): boolean
```

Dirty versions can be identified by an asterisk (*) just after the version name.

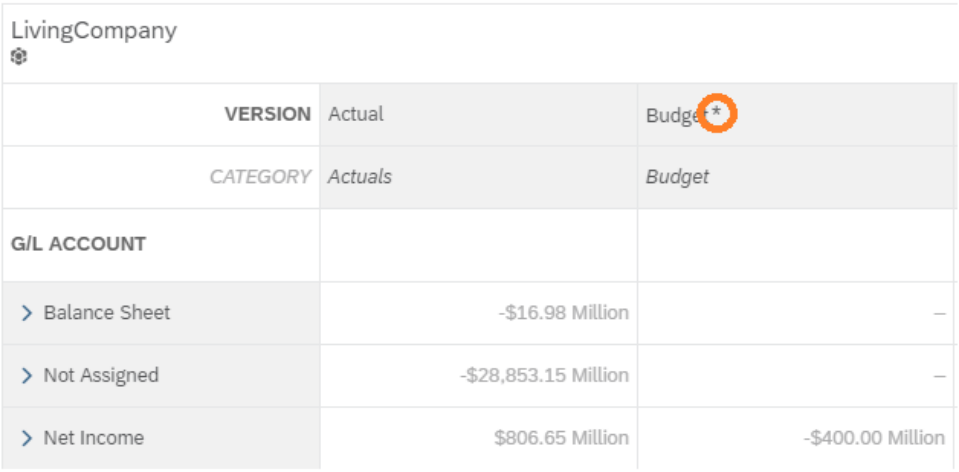


Figure 81: Dirty Version

It's also possible to publish private versions via the two scripting options below:

```
publish(): boolean
publishAs(newVersionName: String, versionCategory: PlanningCategory): boolean
```

In the second option, a version name is given, and a new public version is created under the informed version category.

These scripts can be very useful if your planning model is placed in a popup, for example. As the toolbar is kept in the background Canvas, users don't need to close the popup to then publish the data. With scripting, you can do it directly in the popup!

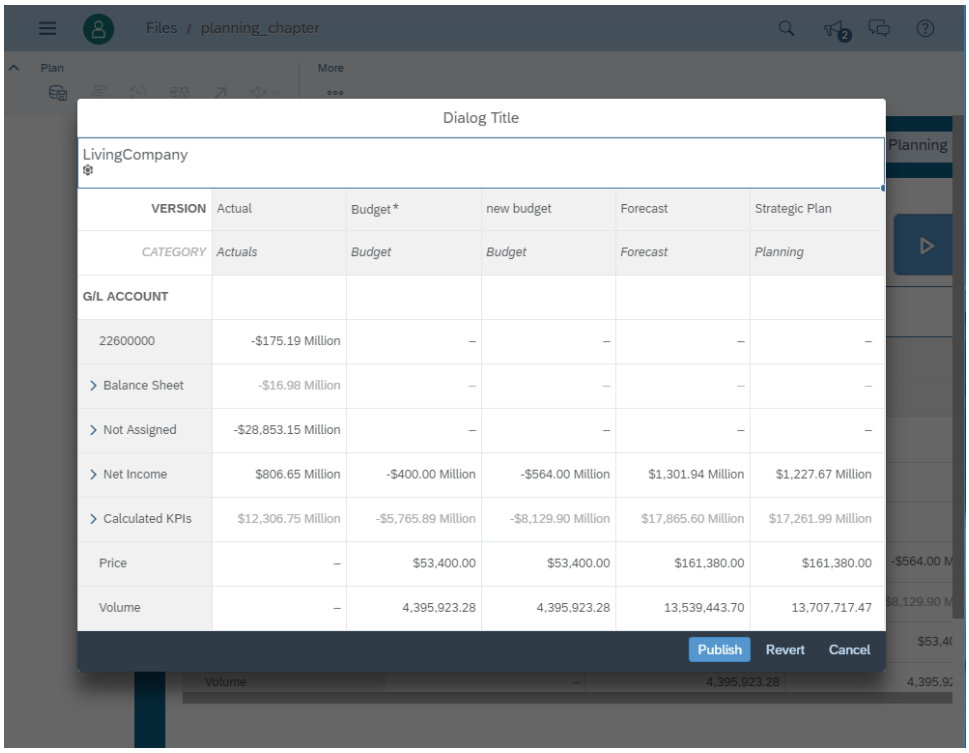
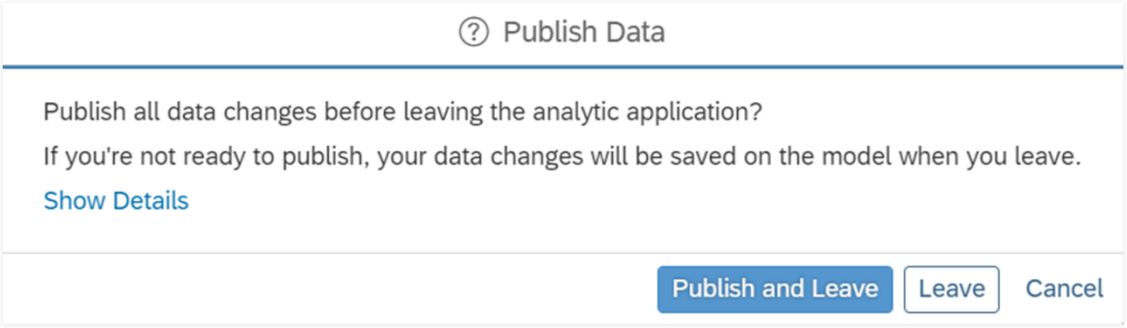


Figure 82: Planning Table in Popup

Find in the next section more information about version category and how to create private versions.

Publish and Leave Dialog for Planning

Before leaving the analytic application, check whether to publish your data changes. You'll also be reminded when leaving the application without publishing:



To not show this dialog at runtime, at design time, go to the *Styling* panel of Canvas, and under *Planning Settings* deselect *Remind of publishing all data changes before leaving*.

7.6.2 Copying

Data models with planning-enabled capability have one dimension in common, the version. Each version is classified into one of the following planning categories:

- Actual
- Planning
- Budget

- Forecast
- Rolling Forecast

The version category 'Actual' is created automatically and can't be deleted.

You can use the `PlanningVersion.copy()` script API method to create a private copy of any version:

```
copy(newVersionName: string, planningCopyOption: PlanningCopyOption,
versionCategory?: PlanningCategory): boolean
```

You can use one of the values of `PlanningCopyOptions` to do the following:

- `PlanningCopyOptions.NoData` – Create a new empty version.
- `PlanningCopyOptions.AllData` – Copy all data from the source version.
- `PlanningCopyOptions.PlanningArea` – Copy only the planning area data from the source version.

7.7 Data Locking

You can use the Data Locking script API to find out if a model is data locking enabled and to set or get the data locking state, even if the table isn't planning-enabled.

The Data Locking script API consists of the following methods:

- `Table.getPlanning().getDataLocking()`
- `Table.getPlanning().getDataLocking.getState()`
- `Table.getPlanning().getDataLocking.setState()`

7.7.1 Using getDataLocking()

You can use `getDataLocking()` to check if a model is data locking enabled.

This check is necessary because a user can't perform certain operations on a table, like `setState()` and `getState()`, if the model isn't data locking enabled.

In the following example, the data locking object is retrieved and printed to the console. A data locking object is returned if data locking is enabled on the model.

```
var planning = Table_1.getPlanning();
console.log(planning.getDataLocking());
```

Note that you can also check if a model is data locking enabled in SAP Analytic Cloud by checking the model preferences (see figure below).

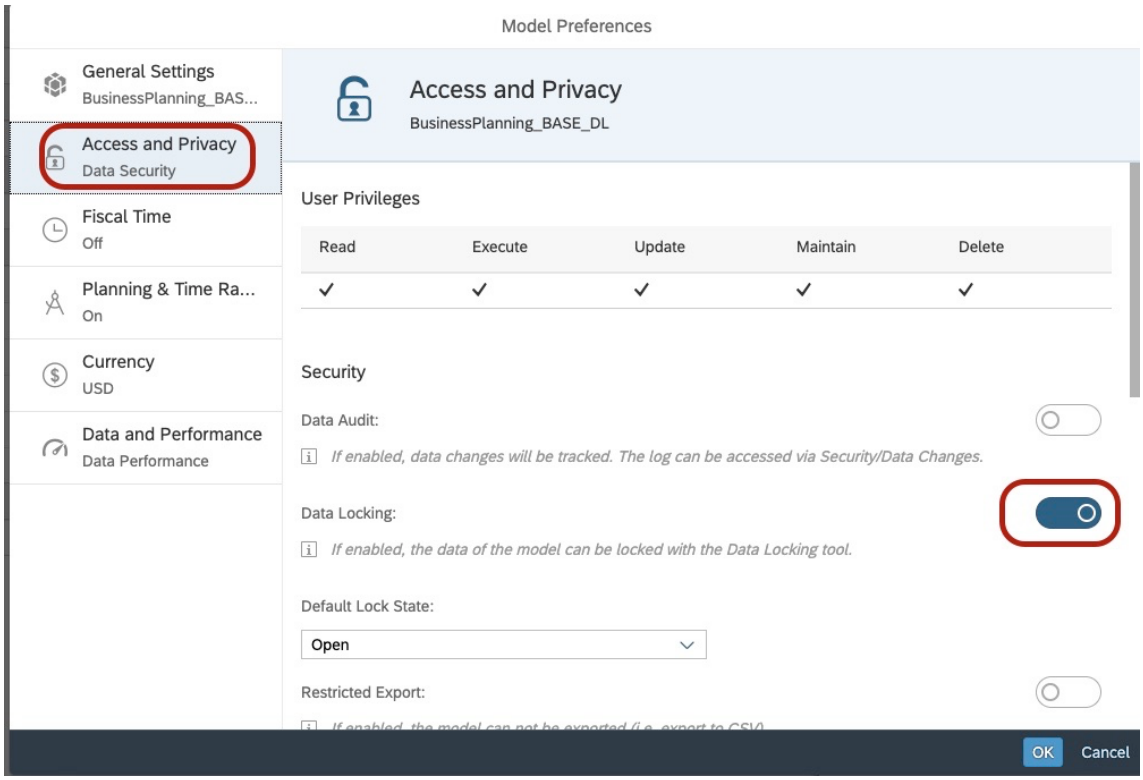


Figure 83: Enabling Data Locking in the Model Preferences

7.7.2 Using getState()

You can use `getState()` to get the data locking state of a cell belonging to a driving dimension. This method works only for SAP Analytics Cloud planning models that are data locking enabled.

In following example, the data locking state for a selected cell of a table is retrieved:

```
var selection = Table_1.getSelections()[0];
var selectionLockState =
Table_1.getPlanning().getDataLocking().getState(selection);
```

In order to create a selection on the table, you can either select the cell in the table manually or you can create the selection string yourself in the script editor.

This method returns one of the following values:

- `DataLockingState.Open`
- `DataLockingState.Restricted`
- `DataLockingState.Locked`
- `DataLockingState.Mixed`

If the state of the selection can't be determined, then the method returns `undefined`. This happens if one of the following situations applies:

- The selection is invalid.
- The cell referenced by the selection isn’t found.
- The cell is in an unknown state.
- The cell has been created using "Add Calculation" at runtime.

If you've activated the *Show Locks* option for the table, then the "lock" icons will be updated after the method has finished running.

7.7.3 Using setState()

You can use `setState()` to set the data locking state of a cell belonging to a driving dimension. This method works only for SAP Analytics Cloud planning models that are data locking enabled.

The method returns `true` if the set operation was successful and `false` otherwise.

You can't set the data locking state on a private version. In this case, the following message is displayed:

"You can only set data locks on public versions. Please use a public version and try again."

You can set one of the following data locking states:

- `DataLockingState.Open`
- `DataLockingState.Restricted`
- `DataLockingState.Locked`

If you attempt to set the data locking state `DataLockingState.Mixed`, then the following message is displayed:

"*You can't set the state with the value 'mixed'. Please specify either 'open', 'restricted' or 'locked' as value."*

The same message is displayed at runtime if you attempt to execute the script and the script fails.

If you select multiple cells and attempt to set the data locking state, the data locking state will be applied to the first selection only.

In the following example, the data locking state is set for a selected table cell:

```
var selection = Table_1.getSelections()[0];
var isSetStateSuccessful =
Table_1.getPlanning().getDataLocking().setState(selection,
DataLockingState.Locked);
```

Note that if data locking is disabled for a model, all locks will be deleted. If it's turned on again later, all members are reset to their default locking state. The same happens if the default locking state or driving dimensions are changed.

7.8 Planning Events

There are two planning-related widgets that provide an event before their action is triggered.

7.8.1 BpcPlanningSequence

onBeforeExecute

```
onBeforeExecute(): boolean
```

Called when the user clicks the BPC planning sequence trigger. If the method returns true or returns no value, then the BPC planning sequence is executed. If the method returns false, then the BPC planning sequence is ignored.

7.8.2 DataActionTrigger

onBeforeExecute

```
onBeforeExecute(): boolean
```

Called when the user clicks the data action trigger. If the method returns true or returns no value, then the data action is executed. If the method returns false, then the data action is ignored.

7.9 Members on the Fly

You can use the PlanningModel script API to add, update, retrieve, and delete members of dimensions of a planning model. You can also update or delete existing members of a planning model, provided you've the appropriate rights.

Example:

In the following example, a new planning member is added to the dimension "LOCATION" of a planning model:

```
PlanningModel_1.createMembers("LOCATION", {
  id: "BERLIN",
  description: "Berlin"
});
```

Note: You can create multiple members with `createMembers()`. When two members have the same member ID then this results in an error.

Example:

In the following example, the new planning member is updated by adding a data locking owner:

```
PlanningModel_1.updateMembers("LOCATION", {
  id: "BERLIN",
  dataLockingOwners: [{
    id: "ADMIN",
    type: UserType.User
  }]
});
```

In the following example, the description of the new planning member is printed to the browser console:

```
var member = PlanningModel_1.getMember("LOCATION", "BERLIN");
console.log(member.description);
```

In the following example, the fifth up to the twelfth member of dimension "LOCATION" is returned:

```
var members = PlanningModel_1.getMembers("LOCATION", {
  offset: "4",
  limit: "8"
});
```

The first member has an offset of 0, so the fifth member has an offset of 4. Starting at this offset, the next 8 members are returned.

The PlanningModel script API provides many more features. For more information, see the online API reference.

Note: You can add members to dimensions of type “Generic” only (see the SAP Analytics Cloud modeler). Adding members to dimension of other types, such as, for example, “Account”, “Version”, “Time”, or “Organization” isn’t supported.

Note: If you’ve added, updated, or deleted members, then call `DataSource.refreshData()` or `Application.refreshData()` if you need the chart or table to reflect the modified members in subsequent method calls operating on visible cells or elements of those widgets, for example, `DataSource.getPlanning().getState()`, `DataSource.getPlanning().setState()`, `DataSource.getData()`, or `Planning.setUserInput()`.

Note: After you’ve added members to a very large model (with millions of members) and have refreshed the model with `Application.refreshData()` or `DataSource.refreshData()`, it may happen that not all added members are immediately displayed, for example, in a table associated with the planning model. The same applies to updating members. This is because these operations work asynchronously in the background. Repeat your refresh operation after a short while.

Note: When retrieving planning model members, represented by `PlanningModelMember` objects, the values of specific properties of these members are only returned if the members’ dimension is configured to provide these values. The following table lists the planning model member property and the right/access that must be granted to this member’s dimension in the SAP Analytics Cloud modeler to receive the property’s value:

Planning Model Member Property	Right/Access
`dataLockingOwner`	Data Locking Ownership
`responsible`	Responsible
`readers`	Data Access Control
`writers`	Data Access Control

Note: When you add your own properties to planning members, use a prefix to avoid name conflicts with existing properties of planning members.