

---

## **AIMMS Function Reference - Case Functions**

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit [www.aimms.com](http://www.aimms.com)

# Case Functions

AIMMS supports the following functions for accessing the cases in the **Data Manager**:

- `CaseCompareIdentifier`
- `CaseCreate`
- `CaseCreateDifferenceFile`
- `CaseDelete`
- `CaseFind`
- `CaseGetChangedStatus`
- `CaseGetDatasetReference`
- `CaseGetType`
- `CaseLoadCurrent`
- `CaseLoadIntoCurrent`
- `CaseMerge`
- `CaseNew`
- `CaseReadFromSingleFile`
- `CaseSave`
- `CaseSaveAll`
- `CaseSaveAs`
- `CaseSelect`
- `CaseSelectMultiple`
- `CaseSelectNew`
- `CaseSetChangedStatus`
- `CaseSetCurrent`
- `CaseWriteToSingleFile`

---

## CaseCompareIdentifier

With the function `CaseCompareIdentifier` you can determine whether or not two cases differ with respect to a certain identifier.

```
CaseCompareIdentifier(  
  FirstCase,  ! (input) element in the set AllCases  
  SecondCase, ! (input) element in the set AllCases  
  Identifier, ! (input) element in the set AllIdentifiers  
  Suffix      ! (optional) element in the set AllSuffixes  
  Mode        ! (optional) element in the set AllCaseComparisonModes  
)
```

### Arguments:

#### *FirstCase*

An element in the set `AllCases`

#### *SecondCase*

An element in the set `AllCases`

#### *Identifier*

An element in the set `AllIdentifiers`, referring to the specific identifier that you want to compare.

#### *Suffix*

An element in the set `AllSuffixes` with respect to which you want to compare the data.

#### *Mode*

An element in the `AllCaseComparisonModes` with respect to how you want to compare the data.

### Return value:

- For numerical identifiers the function returns the differences between the values of the identifier in both cases, based on the mode. It can be the minimum, maximum, average, sum or count of all differences.
- For non-numerical identifiers the function counts the number of differences between the identifier in both cases.

---

## CaseCreate

The procedure `CaseCreate` creates a new case node in the Data Management tree. The name of the case and the folder in which it is created is given as an argument to the function.

```
CaseCreate(  
    case_path,    ! (input) scalar string expression  
    case         ! (output) element parameter into AllCases  
)
```

### Arguments:

*case\_path*

A string expression holding the path and name of the new case. The path is specified relative to the root of the case tree.

*case*

An element parameter into `AllCases`. On successful return this parameter will refer to the newly created element in `AllCases`.

### Return value:

The procedure returns 1 if the case is created successfully. It returns 0 if the case could not be created or if the case already exists.

### Remarks:

- This function is only applicable if the project option Data Management style is set to `Single Data Manager` file.
- If the specified path contains folders that do not exist, then these folders are created automatically. To check whether a specific case path already exists you can use the function `CaseFind`.

### See also:

The procedures [CaseFind](#), [CaseDelete](#).

---

## CaseCreateDifferenceFile

With the procedure `CaseCreateDifferenceFile` you can create an AIMMS input file containing the differences between the current data and a reference case.

```
CaseCreateDifferenceFile(
    referenceCase,      ! (input) element in the set AllCases
    outputFilename,    ! (input) scalar string expression
    diffTypes,         ! (input) indexed element parameter
    absoluteTolerance, ! (optional) scalar expression
    relativeTolerance, ! (optional) scalar expression
    outputPrecision,   ! (optional) scalar expression
    respectDomainCurrentCase ! (optional) scalar expression
)
```

### Arguments:

#### *referenceCase*

An element in the set `AllCases` specifying the case to which the current data should be compared.

#### *outputFilename*

A string expression specifying the name of the file the differences are written to.

#### *diffTypes*

An element parameter indexed over (a subset of) `allIdentifiers` with range the predeclared set `AllDifferencingModes`.

#### *absoluteTolerance*

A scalar expression specifying the absolute tolerance when comparing numerical values. The range of this argument is  $[0, \textit{inf})$ , the default is the value of the option `equality_absolute_tolerance`.

#### *relativeTolerance*

A scalar expression specifying the relative tolerance when comparing numerical values. The range of this argument is  $[0, 1]$ , the default is the value of the option `equality_relative_tolerance`.

#### *outputPrecision*

A scalar expression specifying how many decimals should be printed. The range of the argument is  $\{0 \dots 20\}$ , the default is the value of the option `listing_precision`.

#### *respectDomainCurrentCase*

A scalar expression specifying whether or not the current domain should be taken into account. When 0: The current domain is not taken into account and all differences are written to the output file. When 1: The current domain is taken into account; the differences are filtered according to the domain of the identifier.

**Return value:**

This procedure returns 0 upon failure, 1 upon success. When successful all differences between the current model data and the data in the reference case are written to a file.

---

## CaseDelete

The procedure `CaseDelete` deletes a specific case node from the Data Management tree.

```
CaseDelete(  
    case    ! (input) element parameter into AllCases  
)
```

### Arguments:

*case*

An element parameter into `AllCases`, representing the case that you want to delete.

### Return value:

The procedure returns 1 if the case is deleted successfully, or 0 otherwise.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.

### See also:

The procedure [CaseFind](#).

---

## CaseFind

The procedure CaseFind searches the Data Management tree for a case with a specific name.

```
CaseFind(  
    case_path,    ! (input) scalar string expression  
    case         ! (output) element parameter into AllCases  
)
```

### Arguments:

*case\_path*

A string expression holding the path and name of a case. The path is specified relative to the root of the case tree.

*case*

An element parameter into AllCases. On successful return this parameter will refer to the case found.

### Return value:

The procedure returns 1 if the case is found, and 0 otherwise.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.

### See also:

The procedures [CaseCreate](#), [CaseDelete](#).

---

## CaseGetChangedStatus

The function `CaseGetChangedStatus` returns whether the data of the currently active case has changed and thus needs to be saved.

`CaseGetChangedStatus`

### Arguments:

*None*

### Return value:

The function returns 1 if the data has changed, 0 otherwise.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.

### See also:

The functions [CaseSetChangedStatus](#), [CaseSave](#).

---

## CaseGetDatasetReference

With the function `CaseGetDatasetReference` you can, for every data category, obtain a reference to the dataset that is included in a specific case.

```
CaseGetDatasetReference(  
    case,           ! (input) element from the set AllCases  
    data_category, ! (input) element from the set AllDataCategories  
    dataset        ! (output) element parameter into AllDataSets  
)
```

### Arguments:

#### *case*

An element in the set `AllCases`, referring to the case for which you want to retrieve the dataset reference.

#### *data-category*

An element in the set `AllDataCategories`, referring to the specific data category for which you want to obtain the dataset reference.

#### *dataset*

An element parameter into `AllDataSets`, on return this argument will contain the included dataset. It is set to the empty element if no dataset is included or if the dataset no longer exists.

### Return value:

If any of the first two arguments does not refer to a valid case or data category, or if the data category is not part of the case type, then the function returns `-1` and `CurrentErrorMessage` will contain a proper error message. If a dataset is included, and this dataset still exists, then the function returns `1` and the argument *dataset* will refer to that dataset. If there is no dataset included, then the function returns `1` and *dataset* is set to the empty element. If a dataset is included, but this dataset has been deleted, then the function returns `0` and *dataset* is set to the empty element.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.
- You can use the functions `CaseGetType` and `CaseTypeCategories` to check whether a specific data category is part of a case.

### See also:

The functions [CaseGetType](#), [CaseTypeCategories](#).

---

## CaseGetType

The procedure CaseGetType retrieves the case type for a specific case.

```
CaseGetType(  
    case,           ! (input) element of the set AllCases  
    case_type      ! (output) element parameter into AllCaseTypes  
)
```

### Arguments:

*case*

An element of the set AllCases, referring to the case for which you want to retrieve its case type.

*case\_type*

An element parameter into AllCaseTypes, on successful return this argument will contain the case type for the given case.

### Return value:

The procedure returns 1 on success, 0 otherwise.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.

---

## CaseLoadCurrent

The procedure `CaseLoadCurrent` loads an existing case as the new current case. You can use it to load either a case that is passed as argument to the procedure, or a case that the user can select via a dialog box. If the data of the currently loaded case has changed, then the user is asked to save this data first.

```
CaseLoadCurrent(
    case,      ! (input/output) An element parameter into AllCases
    [dialog],  ! (optional) 0 or 1
    [keepUnreferencedRuntimeLibs ! (optional) 0 or 1
)
```

### Arguments:

*case*

An element parameter into the pre-defined set `AllCases`. If the argument *dialog* is set to 0, then no dialog box is shown and the case to which the element parameter currently refers is loaded. If the argument *dialog* is set to 1, then the value of the element parameter is used to initialize the dialog box. The case that the user has selected and is loaded successfully is returned through this argument.

*dialog (optional)*

An integer value indicating whether or not the user gets a dialog box in which he can select the case to load. The default value is 1, thus if this argument is omitted the dialog box will be shown.

*keepUnreferencedRuntimeLibs (optional)*

An integer value indicating whether or not any runtime libraries in existence before the case is loaded, but not referenced in the case, should be kept in memory or destroyed during the case load. The default is 0 indicating that the runtime libraries not referenced in the case should be destroyed during the case load.

### Return value:

The procedure returns 1 on success. If the user cancelled the operation, then the procedure returns 0. If any other error occurs then the procedure returns -1 and `CurrentErrorMessage` will contain a proper error message.

### Remarks:

- This function is only applicable if the project option `Data Management style` is set to `Single Data Manager file`.
- If you want to suppress the dialog box for the unsaved data, then you may call `CaseSetChangedStatus(0)` prior to `CaseLoadCurrent`.

**See also:**

The procedures [CaseLoadIntoCurrent](#), [CaseMerge](#), [CaseSave](#), [CaseSetChangedStatus](#).

---

## CaseLoadIntoCurrent

The procedure `CaseLoadIntoCurrent` loads the data of an existing case into the current case. You can use it to load either a case that is passed as argument to the procedure, or a case that the user can select via a dialog box. The data that is stored in the case will overwrite any data of the currently active case, and thus this current case is set to have changed data.

```
CaseLoadIntoCurrent(  
  case, ! (input/output) An element parameter into AllCases  
  [dialog] ! (optional) 0 or 1  
  [keepUnreferencedRuntimeLibs ! (optional) 0 or 1  
)
```

### Arguments:

#### *case*

An element parameter into the pre-defined set `AllCases`. If the argument *dialog* is set to 0, then no dialog is shown and the case to which the element parameter currently refers is loaded. If the argument *dialog* is set to 1, then the value of the element parameter is used to initialize the dialog box. The case that the user has selected and is loaded successfully is returned through this argument.

#### *dialog (optional)*

An integer value indicating whether or not the user gets a dialog box in which he can select the case to load. The default value is 1, thus if this argument is omitted the dialog box will be shown.

#### *keepUnreferencedRuntimeLibs (optional)*

An integer value indicating whether or not any runtime libraries in existence before the case is loaded, but not referenced in the case, should be kept in memory or destroyed during the case load. The default is 0 indicating that the runtime libraries not referenced in the case should be destroyed during the case load.

### Return value:

The procedure returns 1 on success. If the user cancelled the operation, then the procedure returns 0. If any other error occurs then the procedure returns -1 and `CurrentErrorMessage` will contain a proper error message.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.

### See also:

The procedures [CaseLoadCurrent](#), [CaseMerge](#), [CaseSave](#), [CaseSetChangedStatus](#).

---

## CaseMerge

The procedure `CaseMerge` merges the data of an existing case with the current data. You can use it to merge either a case that is passed as argument to the procedure, or a case that the user can select via a dialog box. Only the non-default data that is stored in the case will be merged with the data of the currently active case. This current case is set to have changed data.

```
CaseMerge(  
  case, ! (input/output) An element parameter into AllCases  
  [dialog], ! (optional) 0 or 1  
  [keepUnreferencedRuntimeLibs ! (optional) 0 or 1  
)
```

### Arguments:

#### *case*

An element parameter into the pre-defined set `AllCases`. If the argument *dialog* is set to 0, then no dialog box is shown and the case to which the element parameter currently refers is merged. If the argument *dialog* is set to 1, then the value of the element parameter is used to initialize the dialog box. The case that the user has selected and is merged successfully is returned through this argument.

#### *dialog (optional)*

An integer value indicating whether or not the user gets a dialog box in which he can select the case to merge. The default value is 1, thus if this argument is omitted the dialog box will be shown.

#### *keepUnreferencedRuntimeLibs (optional)*

An integer value indicating whether or not any runtime libraries in existence before the case is merged, but not referenced in the case, should be kept in memory or destroyed during the case merge. The default is 1 indicating that the runtime libraries not referenced in the case will be retained during the case merge.

### Return value:

The procedure returns 1 on success. If the user cancelled the operation, then the procedure returns 0. If any other error occurs then the procedure returns -1 and `CurrentErrorMessage` will contain a proper error message.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.

### See also:

The procedures [CaseLoadCurrent](#), [CaseLoadIntoCurrent](#), [CaseSave](#), [CaseGetChangedStatus](#).

---

## CaseNew

The procedure `CaseNew` starts a new case. The procedure is similar to the command **New Case** from the **Data** menu. The procedure does not change any of the current data, it only assures that there is no longer a current case. If you did have a current case and the data of this case has been changed, then AIMMS will ask whether or not this case should be saved first.

`CaseNew`

### Arguments:

*None*

### Return value:

The procedure returns 1 on success. If the user cancelled the operation, then the procedure returns 0. If any other error occurs then the procedure returns -1 and `CurrentErrorMessage` will contain a proper error message.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.
- If you use `CaseNew`, then the name of this new case is not specified until you save the case. If you want to start a new named case, then you can use the following piece of code:

```
if ( CaseGetChangedStatus ) then
  if ( CaseSave = 0 ) then
    return ;
  endif ;
endif ;
if ( CaseSelectNew( a_case ) ) then
  CaseSetCurrent( a_case );
  CaseSetChangedStatus( a_case, 1 );
endif ;
```

### See also:

The procedures [CaseLoadCurrent](#), [CaseSave](#), [CaseSelectNew](#), [CaseSetCurrent](#).

---

## CaseReadFromSingleFile

The procedure `CaseReadFromSingleFile` reads the data from a single case file on disk.

```
CaseReadFromSingleFile(  
  inputFileName          ! (input) scalar string expression  
)
```

### Arguments:

*inputFileName*

A string expression holding the path and name of the input file.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### See also:

The procedures [CaseWriteToSingleFile](#), [CaseSave](#).

---

## CaseSave

The procedure `CaseSave` saves the data to the current case. If there is no current case, then the procedure behaves exactly as the `CaseSaveAs` procedure. If the case has active references to datasets that contain changed data, then these datasets are saved as well.

```
CaseSave(  
    [confirm]      ! (optional) 0, 1 or 2  
)
```

### Arguments:

*confirm (optional)*

An integer to indicate whether or not the procedure should ask for confirmation before overwriting the existing case. If 0, then no confirmation dialog box is shown. If 1 (default), then whether the confirmation dialog box is shown depends on the case type properties. If 2, then always a confirmation dialog box is shown.

### Return value:

The procedure returns 1 if the case is saved successfully. It returns 0 if the user canceled the save operation. If any other error occurs, then the procedure returns -1 and `CurrentErrorMessage` will contain an error message.

### Remarks:

- This function is only applicable if the project option `Data Management style` is set to `Single Data Manager file`.

### See also:

The procedures [CaseSaveAs](#), [CaseSaveAll](#), [CaseLoadCurrent](#), [CaseGetChangedStatus](#).

---

## CaseSaveAll

With the procedure `CaseSaveAll` you can save (via a single call) the current case and all active datasets that need saving.

```
CaseSaveAll(  
    [confirm]      ! (optional) integer value (0, 1 or 2)  
)
```

### Arguments:

*confirm (optional)*

If 0, then cases and datasets are saved without confirmation. If 2, then AIMMS will display a dialog box for the cases and datasets that are about to be saved and ask for confirmation. If 1 (default), then AIMMS will use the properties of the case type and data categories to determine whether a confirmation dialog box should be displayed.

### Return value:

The procedure returns 1 if the user chooses not to save the data or if the user chooses to save the data and the save was executed successfully. It returns 0 if the user cancelled any of the dialog boxes. If any other error occurs then the procedure returns -1 and `CurrentErrorMessage` will contain a proper error message.

### Remarks:

- This function is only applicable if the project option `Data Management style` is set to `Single Data Manager file`.
- This function always returns 1 if the IDE is not loaded, for example when running the component version of AIMMS or when running with the command line option `--as-server`.

### See also:

The procedures [CaseSave](#), [DatasetSave](#).

---

## CaseSaveAs

The procedure `CaseSaveAs` shows a dialog box in which the user can specify a (new) case to which the data is saved. If the case has active references to datasets that contain changed data, then these datasets are saved as well. When saving these datasets the procedure will simply overwrite the current datasets, thus with `CaseSaveAs` you can only change the current case and not any of the current datasets.

```
CaseSaveAs(  
    case           ! (output) element parameter in AllCases  
)
```

### Arguments:

*case*

An element parameter in `AllCases`. On return this parameter will refer to the case that the user selected.

### Return value:

The procedure returns 1 if the case is saved successfully. It returns 0 if the user canceled the save operation. If any other error occurs, then the procedure returns -1 and `CurrentErrorMessage` will contain an error message.

### Remarks:

- This function is only applicable if the project option `Data Management style` is set to `Single Data Manager file`.

### See also:

The procedures [CaseSave](#), [CaseSaveAll](#), [CaseLoadCurrent](#), [CaseGetChangedStatus](#).

---

## CaseSelect

The procedure `CaseSelect` shows a dialog box in which the user can select an existing case.

```
CaseSelect(  
    case,      ! (output) element parameter in AllCases  
    [title]    ! (optional) string expression  
)
```

### Arguments:

*case*

An element parameter in `AllCases`. On return the case will refer to the selected case.

*title (optional)*

A string expression that is used as the title for the dialog box. If this argument is omitted, then a default title is used.

### Return value:

The procedure returns 1 if the user did select a case. If the user presses **Cancel**, then the procedure returns 0.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.

### See also:

The procedure [CaseSelectNew](#).

---

## CaseSelectMultiple

The procedure `CaseSelectMultiple` shows a dialog box in which the user can select a number of cases (and datasets). The selected subset of cases and datasets is stored in the pre-defined set `CurrentCaseSelection`, which is used in the page objects for which the property **Multiple Cases** is set.

```
CaseSelectMultiple(  
    [cases_only]    ! (optional) 0 or 1  
)
```

### Arguments:

*cases\_only (optional)*

This argument controls whether the user can only select cases or can select both datasets and cases. If this argument is omitted, then the default value is 0, which means that both cases and datasets can be selected.

### Return value:

The procedure returns 1 if the user pressed the **OK** button, and 0 if the user pressed **Cancel**.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.

---

## CaseSelectNew

The procedure `CaseSelectNew` shows a dialog box in which the user can select a new case.

```
CaseSelect(  
    case,      ! (output) element parameter in AllCases  
    [title]    ! (optional) string expression  
)
```

### Arguments:

*case*

An element parameter in `AllCases`. On return the case will refer to the selected case.

*title (optional)*

A string expression that is used as the title for the dialog box. If this argument is omitted, then a default title is used.

### Return value:

The procedure returns 1 if the user did select a case. If the user pressed **Cancel**, then the procedure returns 0.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.
- If via this procedure the user creates a new case (i.e. a new case node in the data management tree), then this case does not yet contain any data. The case will only contain data after you explicitly save data to the case.

### See also:

The procedures [CaseSelect](#), [CaseSetCurrent](#), [CaseSave](#).

---

## CaseSetChangedStatus

The procedure `CaseSetChangedStatus` can set the status of the current case to either changed or unchanged.

```
CaseSetChangedStatus(  
    status,           ! (input) 0 or 1  
    [include_datasets] ! (optional) 0 or 1  
)
```

### Arguments:

#### *status*

An integer value holding the new case status: 0 for unchanged, 1 for changed.

#### *include\_datasets* (optional)

An integer to indicate whether or not the the status of the included and active datasets should be set as well. If you omit this argument, then the default value is 0 (status of datasets is not set).

### Return value:

The procedure returns 1.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.

### See also:

The procedures [CaseGetChangedStatus](#), [DatasetSetChangedStatus](#).

---

## CaseSetCurrent

The procedure `CaseSetCurrent` sets the case that is regarded as the current case. It does not load or save any data or checks whether data needs to be saved. You can, for example, use it to make a newly created case the current case, so that during a `CaseSave` the data is written to this case.

```
CaseSetCurrent(  
    case           ! (input) element of the set AllCases  
)
```

### Arguments:

*case*

An element of the set `AllCases`, referring to the case that should become the current case.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### Remarks:

- This function is only applicable if the project option `Data Management style` is set to `Single Data Manager file`.

### See also:

The procedures [CaseNew](#), [CaseCreate](#), [CaseSelectNew](#), [CaseSave](#).

---

## CaseWriteToSingleFile

The procedure `CaseWriteToSingleFile` writes the current data to a case file on disk.

```
CaseWriteToSingleFile(  
    outputFileName          ! (input) scalar string expression  
)
```

### Arguments:

*outputFileName*

A string expression holding the path and name of the output file.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### Remarks:

- This function is only applicable if the project option Data Management style is set to Single Data Manager file.
- The procedure `CaseWriteToSingleFile` uses the current case type to determine which data should be written. This is usually the case type of the last loaded case. If you want to make sure that a specific case type is used, you can preset the case type via the predefined element parameter `CurrentDefaultCaseType`.

The files written by `CaseWriteToSingleFile` can only be read by `CaseReadFromSingleFile`.

### See also:

The procedures [CaseReadFromSingleFile](#), [CaseSave](#).