

---

## **AIMMS Function Reference - Page 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)

# Page Functions

AIMMS supports the following functions for opening, closing, and manipulating the pages in the interface:

- PageClose
- PageCopyTableToClipboard
- PageCopyTableToExcel
- PageGetActive
- PageGetAll
- PageGetChild
- PageGetFocus
- PageGetNext
- PageGetNextInTreeWalk
- PageGetParent
- PageGetPrevious
- PageGetTitle
- PageGetUsedIdentifiers
- PageOpen
- PageOpenSingle
- PageRefreshAll
- PageSetCursor
- PageSetFocus
- PivotTableDeleteState
- PivotTableReloadState
- PivotTableSaveState
- PrintEndReport
- PrintPage
- PrintPageCount
- PrintStartReport

---

## PageClose

With the procedure `PageClose` you can close a page that is currently open.

```
PageClose(  
    page          ! (optional) string expression  
)
```

### Arguments:

*page (optional)*

A string expression representing the name of the page that you want to close. This name is the unique name as it appears in the Page Manager tree. If you omit this argument, then `PageClose` closes the currently active page.

### Return value:

The procedure returns 1 if the page is closed successfully, or a 0 otherwise.

### See also:

The procedures [PageOpen](#), [PageOpenSingle](#).

---

## PageCopyTableToClipboard

With the procedure `PageCopyTableToClipboard` you can copy (part of) a specific table on a specific page to the clipboard, so that you subsequently can paste it in any other application.

```
PageCopyTableToClipboard(  
    pageName,      ! (input) scalar string expression  
    tag,           ! (input) scalar string expression  
    includeHeaders, ! (input) scalar numerical expression  
    selectionOnly  ! (input) scalar numerical expression  
)
```

### Arguments:

#### *pageName*

A string expression representing the name of the page containing the table.

#### *tag*

A string expression representing the tag name of the table for which you want to copy the current displayed data. This can be a Composite Table, a Pivot Table or an standard Table object.

#### *includeHeaders*

A scalar numerical expression to control whether or not the headers should be copied as well. If `includeHeaders` is not equal to 0 then the headers are included.

#### *selectionOnly*

A scalar numerical expression to control whether the entire table or only the currently selected cells should be copied. If `selectionOnly` is not equal to 0 then only the currently selected cells (with or without the corresponding headers, based on the value of `includeHeaders`) are copied.

### Return value:

The procedure returns 1 on success. If it fails, then it returns 0 and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

### Remarks:

You can specify a unique tag name for each page object via the object properties.

### See also:

The procedure [PageCopyTableToExcel](#).

---

## PageCopyTableToExcel

With the procedure `PageCopyTableToExcel` you can copy (part of) a specific table on a specific page directly to a range in Excel.

```
PageCopyTableToExcel(
    pageName,      ! (input) scalar string expression
    tag,           ! (input) scalar string expression
    includeHeaders, ! (input) scalar numerical expression
    selectionOnly, ! (input) scalar numerical expression
    ExcelWorkbook, ! (input) scalar string expression
    Range,        ! (input) scalar string expression
    [Sheet]       ! (optional) scalar string expression
)
```

### Arguments:

#### *pageName*

A string expression representing the name of the page containing the table.

#### *tag*

A string expression representing the tag name of the table for which you want to copy the current displayed data. This can be a Composite Table, a Pivot Table or an standard Table object.

#### *includeHeaders*

A scalar numerical expression to control whether or not the headers should be copied as well. If `includeHeaders` is not equal to 0 then the headers are included.

#### *selectionOnly*

A scalar numerical expression to control whether the entire table or only the currently selected cells should be copied. If `selectionOnly` is not equal to 0 then only the currently selected cells (with or without the corresponding headers, based on the value of `includeHeaders`) are copied.

#### *ExcelWorkbook*

A scalar string expression representing the Excel workbook.

#### *Range*

A scalar string expression containing the (named) range in the Excel sheet to which the table should be copied.

#### *Sheet*

The sheet to which the table should be copied. Default is the active sheet.

### Return value:

The procedure returns 1 on success. If it fails, then it returns 0 and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

**Remarks:**

- By calling the procedure `ExcelSetActiveSheet` you can set the active sheet, after which the optional sheet argument can be omitted in procedures like this one.
- A call to this procedure with a specified sheet argument does not change the active sheet, except when the workbook does not have an active sheet yet.
- When the dimensions of the specified range do not match the dimensions of the table on the clipboard, then the standard Excel rules for pasting are applied. That is:
  - if the range is only one column wide, then the range will automatically be expanded horizontally to match the number of columns on the clipboard,
  - else if the number of columns in the range is smaller than the number of columns on the clipboard then only the first columns that fit will be copied,
  - else if the number of columns in the range is larger than the number of columns on the clipboard, the range is made smaller.

A similar algorithm is used for the number of rows. So if you want to make sure that the entire contents of the copied table is pasted in Excel, you can best specify a range of exactly one cell.

- You can specify a unique tag name for each page object via the object properties.

**See also:**

The procedure [PageCopyTableToClipboard](#).

---

## PageGetActive

With the procedure `PageGetActive` you can retrieve the name of the currently active page.

```
PageGetActive(  
    page          ! (output) scalar string identifier  
)
```

### Arguments:

*page*

A string identifier to hold the name of the page that is currently active. If the same page name is used in more than one (library) project, then the prefix of the library project (or `::` in case of the main project) will be prepended.

### Return value:

The procedure returns 1 on success, or 0 if there is no currently active page.

### See also:

The procedure [PageGetFocus](#).

---

## PageGetAll

With the procedure `PageGetAll` you can retrieve the names of all pages and/or templates in your project

```
PageGetAll(  
    page_set,          ! (output) an (empty) root set  
    IncludePages,     ! (optional, default 1) scalar expression  
    IncludeTemplates, ! (optional, default 1) scalar expression  
    ExcludeHidden,    ! (optional, default 0) scalar expression  
    ExcludePrintables ! (optional, default 0) scalar expression  
)
```

### Arguments:

#### *page\_set*

A root set, that on return will contain the names of all the requested pages.

#### *IncludePages*

A scalar numerical expression to indicate whether the returned set should contain the names of pages in your project.

#### *IncludeTemplates*

A scalar numerical expression to indicate whether the returned set should contain the names of templates in your project.

#### *ExcludeHidden*

A scalar numerical expression to indicate whether hidden pages should be part of the returned set. If `ExcludeHidden` is set to 1 then the returned set will not contain any page that is currently hidden.

#### *ExcludePrintables*

A scalar numerical expression to indicate whether print pages or print templates should be part of the returned set. Print pages/templates are those pages/templates that are especially created for printing (i.e. in the Template Manager they are placed as children of a root print template). If `ExcludePrintables` is set to 1 then the returned set will not contain any printable page or template.

### Return value:

The procedure returns 1 on success, and 0 on failure.

### See also:

The procedures [PageGetNext](#), [PageGetPrevious](#), [PageGetChild](#), [PageGetParent](#), [PageGetNextInTreeWalk](#).

---

## PageGetChild

The procedure `PageGetChild` retrieves the name of the first child page for a specific page in the Page Manager tree.

```
PageGetChild(  
    page,           ! (input) scalar string expression  
    childpage,     ! (output) scalar string identifier  
    IncludeHiddenPages ! (optional) scalar numerical expression  
)
```

### Arguments:

#### *page*

A string expression containing the name of a (parent) page in the Page Manager tree.

#### *childpage*

A scalar string identifier to hold the name of the first child page beneath the given parent page (if any).

#### *IncludeHiddenPages*

A scalar numerical expression to indicate whether hidden pages should be taken into account. If `IncludeHiddenPages` is set to 1 then the resulting child page may be a page that is currently hidden, otherwise these hidden pages are skipped. The default is 0.

### Return value:

The procedure returns 1 on success, or 0 if the given page name does not exist or if the page does not have any child pages.

### See also:

The procedures [PageGetParent](#), [PageGetNext](#), [PageGetPrevious](#), [PageGetNextInTreeWalk](#), [PageGetAll](#).

---

## PageGetFocus

With the procedure `PageGetFocus` you can retrieve the name of the currently active page.

```
PageGetFocus(
    page,          ! (output) scalar string identifier
    tag,          ! (output) scalar string identifier
    [fullPathTag] ! (optional) 0 or 1
)
```

### Arguments:

*page*

A string identifier to hold the name of the currently active page. If the same page name is used in more than one (library) project, then the prefix of the library project (or `::` in case of the main project) will be prepended.

*tag*

A string identifier to hold the tag name of the object that currently has the keyboard input focus.

*fullPathTag (optional)*

If this value is set to 0, then returned tag will be the simple tag name of the object that has focus. If this value is set to 1 (the default), then the returned tag name will also contain the tags of Tabbed or Indexed Page objects in which the object with focus is contained. See the remarks below.

### Return value:

The procedure returns 1 on success, or 0 if there is no currently active page or if no object has the input focus.

### Remarks:

You can specify a unique tag name for each page object via the object properties. If no tag name has been given explicitly, then the type of object is returned (“Table”, “Bar Chart”, etc.)

If an object with tag “X” is displayed in a tabbed page object with tag “T”, then the full path tag name will be “T::X”.

If an object with tag “X” is displayed in an indexed page object with tag “IP” on a row and column that corresponds with elements “rowi” and “colj”, then the full path tag name will be “IP(‘rowi’,‘colj’)::X”.

### See also:

The procedures [PageSetFocus](#), [PageGetActive](#).

---

## PageGetNext

The procedure `PageGetNext` retrieves the name of the next page for a specific page in the Page Manager tree. The next page is the page that has the same parent page, and is positioned directly below the given page.

```
PageGetNext(  
    page,           ! (input) scalar string expression  
    nextpage,      ! (output) scalar string identifier  
    IncludeHiddenPages ! (optional) scalar numerical expression  
)
```

### Arguments:

#### *page*

A string expression containing the name of a (child) page in the Page Manager tree.

#### *nextpage*

A scalar string identifier to hold the name of the next page of the given page (if it exists).

#### *IncludeHiddenPages*

A scalar numerical expression to indicate whether hidden pages should be taken into account. If `IncludeHiddenPages` is set to 1 then the resulting page may be a page that is currently hidden, otherwise these hidden pages are skipped. The default is 0.

### Return value:

The procedure returns 1 on success, or 0 if the given page name does not exist or if the page does not have a next page.

### See also:

The procedures [PageGetPrevious](#), [PageGetChild](#), [PageGetParent](#), [PageGetNextInTreeWalk](#), [PageGetAll](#).

---

## PageGetNextInTreeWalk

The procedure `PageGetNextInTreeWalk` retrieves the name of the next page for a specific page in the Page Manager tree by traversing the tree in a depth-first manner: This procedure will try to find the next page of a page first by searching for child nodes of the selected page. If the page has no child nodes, it will look for a next page on the same level. If there also isn't a next page in the same level, it will try to find a next page for the parent nodes. This procedure includes hidden pages and ignores separators.

```
PageGetNextInTreeWalk(  
    page,           ! (input) scalar string expression  
    nextpage,      ! (output) scalar string identifier  
    IncludeHiddenPages ! (optional) scalar numerical expression  
)
```

### Arguments:

#### *page*

A string expression containing the name of a (child) page in the Page Manager tree.

#### *nextpage*

A scalar string identifier to hold the name of the next page of the given page (if it exists).

#### *IncludeHiddenPages*

A scalar numerical expression to indicate whether hidden pages should be taken into account. If `IncludeHiddenPages` is set to 1 then the resulting parent page may be a page that is currently hidden, otherwise these hidden pages are skipped. The default is 0.

### Return value:

The procedure returns 1 on success, or 0 if the given page name does not exist or if the page does not have a next page.

### See also:

The procedures [PageGetNext](#), [PageGetPrevious](#), [PageGetChild](#), [PageGetParent](#), [PageGetAll](#).

---

## PageGetParent

The procedure `PageGetParent` retrieves the name of the parent page for a specific page in the Page Manager tree.

```
PageGetParent(  
    page,                ! (input) scalar string expression  
    parentpage,         ! (output) scalar string identifier  
    IncludeHiddenPages ! (optional) scalar numerical expression  
)
```

### Arguments:

#### *page*

A string expression containing the name of a (child) page in the Page Manager tree.

#### *parentpage*

A scalar string identifier to hold the name of the parent page of the given page (if it exists).

#### *IncludeHiddenPages*

A scalar numerical expression to indicate whether hidden pages should be taken into account. If `IncludeHiddenPages` is set to 1 then the resulting parent page may be a page that is currently hidden, otherwise these hidden pages are skipped. The default is 0.

### Return value:

The procedure returns 1 on success, or 0 if the given page name does not exist or if the page does not have a parent page.

### See also:

The procedures [PageGetChild](#), [PageGetNext](#), [PageGetPrevious](#), [PageGetNextInTreeWalk](#), [PageGetAll](#).

---

## PageGetPrevious

The procedure `PageGetPrevious` retrieves the name of the previous page for a specific page in the Page Manager tree. The previous page is the page that has the same parent page, and is positioned directly above the given page.

```
PageGetPrevious(  
    page,                ! (input) scalar string expression  
    previouspage,        ! (output) scalar string identifier  
    IncludeHiddenPages ! (optional) scalar numerical expression  
)
```

### Arguments:

*page*

A string expression containing the name of a (child) page in the Page Manager tree.

*previouspage*

A scalar string identifier to hold the name of the previous page of the given page (if it exists).

*IncludeHiddenPages*

A scalar numerical expression to indicate whether hidden pages should be taken into account. If `IncludeHiddenPages` is set to 1 then the resulting page may be a page that is currently hidden, otherwise these hidden pages are skipped. The default is 0.

### Return value:

The procedure returns 1 on success, or 0 if the given page name does not exist or if the page does not have a previous page.

### See also:

The procedures [PageGetNext](#), [PageGetChild](#), [PageGetParent](#), [PageGetNextInTreeWalk](#), [PageGetAll](#).

---

## PageGetTitle

The procedure `PageGetTitle` retrieves the title of a specific page in the Page Manager tree.

```
PageGetTitle(  
    pageName,      ! (input) scalar string expression  
    pageTitle      ! (output) scalar string identifier  
)
```

### Arguments:

*pageName*

A string expression containing the name of a page in the Page Manager tree.

*pageTitle*

A scalar string identifier to hold the title of the given page.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

---

## PageGetUsedIdentifiers

The procedure `PageGetUsedIdentifiers` returns a subset of `AllIdentifiers` containing all identifiers used on a specified page.

```
PageGetUsedIdentifiers(  
    page,          ! (input) scalar string expression  
    identifier_set ! (output) subset of all identifiers  
)
```

### Arguments:

*page*

A string expression containing the name of a page in the Page Manager tree.

*identifier\_set*

A subset of all identifiers containing all the identifiers used in the page.

### Return value:

The procedure returns 1 on success, or 0 if the given page name does not exist.

### See also:

The procedure [IdentifierGetUsedInformation](#).

---

## PageOpen

With the procedure `PageOpen` you can open any page that is defined in the Page Manager. If the page is already open, then the procedure will make this page the active page. The `PageOpen` procedure does not halt the execution, unless the page to open is defined as a dialog page. In the latter case, the execution is halted until the user closes the page.

```
PageOpen(  
    page           ! (input) string expression  
)
```

### Arguments:

*page*

A string expression representing the name of the page that you want to open. This name is the unique name as it appears in the Page Manager tree.

### Return value:

The procedure returns 1 if the page is opened successfully. If the procedure fails to open the page it returns 0, and the pre-defined parameter `CurrentErrorMessage` will contain a proper error message.

### See also:

The procedures [PageOpenSingle](#), [PageClose](#).

---

## PageOpenSingle

The procedure `PageOpenSingle` is similar to `PageOpen`, except that after successful opening the page `PageOpenSingle` makes sure that all other currently opened pages are closed.

```
PageOpenSingle(  
    page          ! (input) string expression  
)
```

### Arguments:

*page*

A string expression representing the name of the page that you want to open. This name is the unique name as it appears in the Page Manager tree.

### Return value:

The procedure returns 1 if the page is opened successfully. If the procedure fails to open the page it returns 0, and the pre-defined parameter `CurrentErrorMessage` will contain a proper error message.

### See also:

The procedures [PageOpen](#), [PageClose](#).

---

## PageRefreshAll

Normally, the data on all open pages is refreshed automatically each time AIMMS has finished executing a procedure. Via a call to `PageRefreshAll` you can refresh the data on all pages at any time during a procedure run (for example to show intermediate results).

`PageRefreshAll`

### Arguments:

*None*

### Remarks:

Pages that you open from within a procedure will always show the data that is available at that moment, so it is not necessary to call `PageRefreshAll` for a newly opened page.

### See also:

The procedure [PageOpen](#).

---

## PageSetCursor

With the procedure `PageSetCursor` you have maximum control over where you want to set the current keyboard input focus. Similar to `PageSetFocus` you can specify which page object should get the focus, but additionally you can specify the data element that should be highlighted within the focus object.

```
PageSetCursor(  
    page           ! (input) scalar string expression  
    tag,           ! (input) scalar string expression  
    scalar_reference, ! (input) scalar identifier  
)
```

### Arguments:

#### *page*

A string expression representing the name of the page in which you want to set the input focus.

#### *tag*

A string expression representing the tag name of the object that should get the keyboard input focus.

#### *scalar\_reference*

A scalar data element that matches the element that you want to highlight within the object.

### Return value:

The procedure returns 1 on success. If it fails, then it returns 0 and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

### Examples:

If you are displaying a variable `Transport` in a table with tag "Transport-Table" on page "Results", then you can set the focus and cursor to a specific cell in this table using the following procedure call:

```
PageSetCursor("Results", "TransportTable", Transport('Amsterdam', 'Rotterdam'));
```

### Remarks:

You can specify a unique tag name for each page object via the object properties.

### See also:

The procedure [PageSetFocus](#).

---

## PageSetFocus

With the procedure `PageSetFocus` you can set the keyboard input focus to a specific object within a specific page. If the page is not open, then the procedure will first try to open the page.

```
PageSetFocus(  
    page,      ! (input) scalar string expression  
    tag        ! (input) scalar string expression  
)
```

### Arguments:

*page*

A string expression representing the name of the page in which you want to set the input focus.

*tag*

A string expression representing the tag name of the object that should get the keyboard input focus.

### Return value:

The procedure returns 1 on success. If it fails to set the focus to the specified object, then the return value is 0 and `CurrentErrorMessage` will contain a proper error message.

### Remarks:

You can specify a unique tag name for each page object via the object properties.

### See also:

The procedures [PageSetCursor](#), [PageGetFocus](#).

---

## PivotTableDeleteState

With the procedure `PivotTableDeleteState` you can delete a specific state in either the Developer or End User state file.

```
PivotTableDeleteState(  
    statename, ! (input) scalar string expression  
    statesource ! (input) scalar string expression  
)
```

### Arguments:

*statename*

A string expression representing the name of the state to be deleted.

*statesource*

A string expression representing the type of state to be deleted. Possible values are:

- `DeveloperState`: Delete the specified state from the *developer* state file.
- `UserState`: Delete the specified state from the *user* state file.
- `Both`: Delete the state from both the *developer* and *user* state file

### Return value:

The procedure returns 1 on success. If it fails to delete the specified state, then the return value is 0 and `CurrentErrorMessage` will contain a proper error message.

### Remarks:

- When running in End User mode, you cannot delete states from the developer state file.

### See also:

- The Pivot Table example that comes with the AIMMS installation includes a library that uses this new function. It includes a right-mouse menu that can be assigned to a Pivot Table, after which the user can save, load, or delete states for that Pivot Table. You can include this library in your own project as well.
- The functions `PivotTableReloadState`, `PivotTableSaveState`.

---

## PivotTableReloadState

With the procedure `PivotTableReloadState` you can reload the state of a specific pivot table from either the developer or user state file.

```
PivotTableReloadState(  
    page,      ! (input) scalar string expression  
    tag,       ! (input) scalar string expression  
    statesource ! (input) scalar string expression  
)
```

### Arguments:

*page*

A string expression representing the name of the page that contains the pivot table.

*tag*

A string expression representing the tag that identifies the pivot table.

*statesource*

A string expression representing the type of state to be reloaded. Possible values are:

- `DeveloperState`: Reload the pivot table with a state that is present in the *developer* state file.
- `UserState`: Reload the pivot table with a state that is present in the *user* state file.
- `None`: Reload the pivot table as if no state was available.

### Return value:

The procedure returns 1 on success. If it fails to reload the state for the specified object, then the return value is 0 and `CurrentErrorMessage` will contain a proper error message.

### Remarks:

- You can specify a unique tag name for each page object on the **Misc** tab of the object properties dialog box.
- The name of the state is specified by the *Specific State Name* property on the **General** tab of the pivot table properties dialog box.
- This procedure will only reload the state when the **Save Layout/State - By Developer** property (or **Save Layout/State - By End User** when running in end-user mode) on the **general** tab of the pivot table properties dialog box, has been set to a value other than *No*.

**See also:**

- The Pivot Table example that comes with the AIMMS installation includes a library that uses this new function. It includes a right-mouse menu that can be assigned to a Pivot Table, after which the user can save, load, or delete states for that Pivot Table. You can include this library in your own project as well.
- The functions `PivotTableSaveState`, `PivotTableDeleteState`.

---

## PivotTableSaveState

With the procedure `PivotTableSaveState` you can save the state of a specific pivot table to either the developer or user state file.

```
PivotTableSaveState(  
    page,      ! (input) scalar string expression  
    tag,       ! (input) scalar string expression  
    statesource ! (input) scalar string expression  
)
```

### Arguments:

*page*

A string expression representing the name of the page that contains the pivot table.

*tag*

A string expression representing the tag that identifies the pivot table.

*statesource*

A string expression representing the type of state to be saved. Possible values are:

- `DeveloperState`: Save the specified state to the *developer* state file.
- `UserState`: Save the specified state to the *user* state file.

### Return value:

The procedure returns 1 on success. If it fails to save the state for the specified object, then the return value is 0 and `CurrentErrorMessage` will contain a proper error message.

### Remarks:

- When running in end-user mode, it is not possible to save a *developer* state.
- You can specify a unique tag name for each page object on the **Misc** tab of the object properties dialog box.
- The name of the state is specified by the *Specific State Name* property on the **General** tab of the pivot table properties dialog box.
- This procedure will only save the state when the **Save Layout/State - By Developer** property (or **Save Layout/State - By End User** when running in end-user mode) on the **general** tab of the pivot table properties dialog box, has been set to a value other than *No*.

**See also:**

- The Pivot Table example that comes with the AIMMS installation includes a library that uses this new function. It includes a right-mouse menu that can be assigned to a Pivot Table, after which the user can save, load, or delete states for that Pivot Table. You can include this library in your own project as well.
- The functions `PivotTableDeleteState`, `PivotTableReloadState`.

---

## PrintEndReport

With the procedure `PageEndReport` you finish the printing of a report that was started via a call to `PrintStartReport`.

`PrintEndReport`

### Arguments:

*None*

### Return value:

The procedure returns 1 on success, or 0, if there was no current report.

### See also:

The procedures [PrintStartReport](#), [PrintPage](#).

---

## PrintPage

With the procedure PrintPage you can print a single print page. If the page contains a data object for which the available data does not fit onto a single printed sheet, AIMMS will print as many sheets as needed.

```
PrintPage(  
    page,                ! (input) scalar string expression  
    [filename,]          ! (optional) scalar string expression  
    [from_pagenr,]      ! (optional) integer  
    [to_pagenr,]        ! (optional) integer  
    [UseDefaultBitmapPrintSettings] ! (optional) integer  
)
```

### Arguments:

#### *page*

A string expression representing the name of the page that you want to print. This name is the unique name as it appears in the Page Manager tree.

#### *filename (optional)*

If this file name is specified, then AIMMS will print to the specific file and not directly to the printer. If this argument is omitted, then AIMMS will print according to the settings of the currently selected printer.

#### *from\_pagenr (optional)*

If the objects on the page result in multiple printed sheets, then with this argument you can specify the first sheet to print. If omitted, then printing will start at the first sheet (*from\_pagenr* = 1).

#### *to\_pagenr (optional)*

If the objects on the page result in multiple printed sheets, then with this argument you can specify the last sheet to print. If omitted, then printing continues until the last sheet.

#### *UseDefaultBitmapPrintSettings (optional)*

When printing a non-print page, the page is printed by creating an exact bitmap copy of the page as it appears on the screen. By default (if the argument equals 0), a dialog will appear in which you can specify which scale should be applied such that it fits on one or more sheets. By settings this argument to 1, this dialog box will be skipped and the bitmap print will use the standard settings of the dialog box. If the page to print is designed as a print page, then this argument is ignored.

### Return value:

The procedure returns the actual number of pages printed if the print page is printed successfully. If the procedure fails to print the page it returns

0, and the pre-defined parameter `CurrentErrorMessage` will contain a proper error message.

**See also:**

The procedures [PrintPageCount](#), [PrintStartReport](#).

---

## PrintPageCount

The procedure `PrintPageCount` will return how many sheets of paper are needed to print a single print page in the interface.

```
PrintPageCount(  
    page          ! (input) scalar string expression  
)
```

### Arguments:

*page*

A string expression representing the name of the page that you want to print. This name is the unique name as it appears in the Page Manager tree.

### Return value:

The procedure returns the number of sheets needed, or 0 if the page cannot be printed.

### See also:

The procedure [PrintPage](#).

---

## PrintStartReport

With the procedure `PrintStartReport` you start printing a report that consists of the printing of multiple pages (using the procedure `PrintPage`). The advantage of printing in the form of a report is that all print request until `PrintEndReport` arrive at the printer as a single print job, and that the pages are numbered correctly.

```
PrintStartReport(  
    title,          ! (input) scalar string expression  
    [filename]     ! (optional) scalar string expression  
)
```

### Arguments:

#### *title*

A string expression representing the title of the report. This title is used in the communication to the printer as the name of the print job.

#### *filename (optional)*

If this file name is specified, then AIMMS will print to the specific file and not directly to the printer. If this argument is omitted, then AIMMS will print according to the settings of the currently selected printer.

### Return value:

The procedure returns 1 on success. If the procedure fails, then the pre-defined parameter `CurrentErrorMessage` will contain a proper error message.

### Remarks:

A successful call to `PrintStartReport` must be followed by a call to `PrintEndReport`, otherwise nothing is printed, and your printer may hang.

### See also:

The procedures [PrintEndReport](#), [PrintPage](#).