Data Checker Utility

User Guide

www.aveva.com

First published September 2007

© AVEVA Solutions Ltd, and its subsidiaries

AVEVA Solutions Ltd, High Cross, Madingley Road, Cambridge, CB3 0HB, United Kingdom

Data Checker Utility User Guide

Contents **Page**

1 Introducing the Data Checker Utility

1.1 Scope of this Guide

This guide describes the Data Checker utility, which allows data consistency checks written in PML to be added to Outfitting Design quickly and easily.

The Checker utility provides:

- A standard user interface for data consistency checks written in PML.
- Standard reporting, to screen and file, of the results of a data consistency check.
- The ability to add new checking functions tailored to company or project working practices.
- Navigation in the database to elements that have failed the checks, by simply picking on part of the on-screen report.

You can add your own customised check functions to any standard data consistency checks provided by AVEVA. A check function is a PML function which tests selected elements in the database, and reports back to the Checker utility whether the element has passed or failed the tests.

This guide does *not* describe how to program in PML. Before reading the customisation chapters of this manual you should be familiar with PML facilities, which are described in full in the *Software Customisation Guide*.

1.2 How the Guide is Organised

- *Running the Data Checker Utility* describes how to use the Checker utility to run predefined data checks.
- *Adding a New Check Function* explains, with some examples, how to add new Check functions. It shows the conventions used when programming Check functions in PML.
- *Modifying and Deleting Check Functions* shows how to modify or delete existing Check functions.
- *Checker Objects* defines the principal PML objects used by the Checker utility.

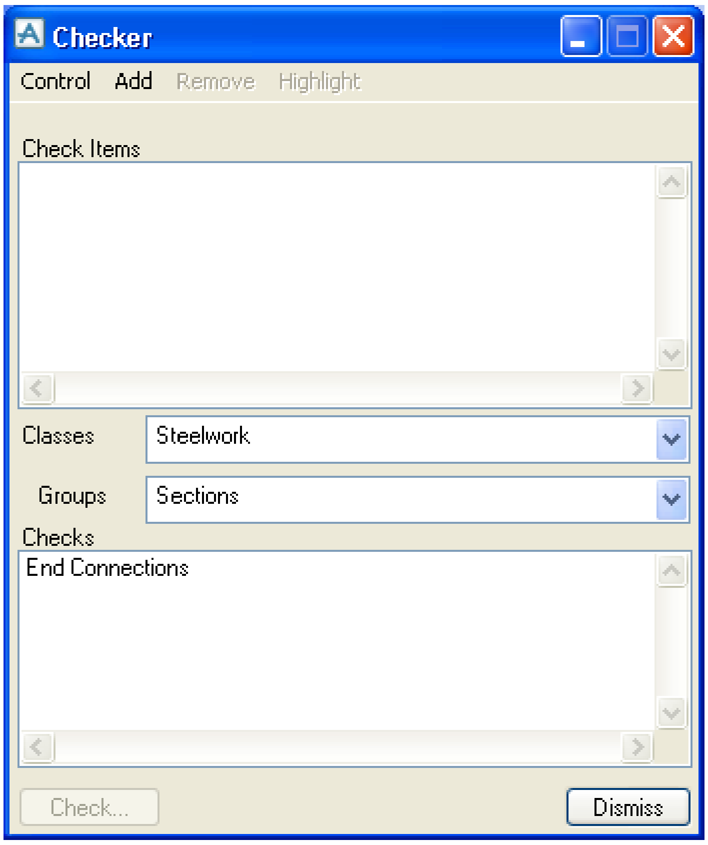
Note: If you simply want to run predefined checks on your design data, you need only refer to *Running the Data Checker Utility*.

If you want to customise the Checker utility by editing the files defining available Check functions, you will need to read the whole of the guide.

2 Running the Data Checker Utility

To access the Data Checker utility from within any Outfitting Design application, select **Utilities>Data Checker** from the application's main menu bar.

All checking operations are controlled from the resulting form:



Note: When this form is first displayed, the **Check Items** list will be empty.

The upper part of the form lets you specify which design elements you want to check; the lower part lets you select one or more check functions to be applied to those elements.

To run a data check, carry out the operations described in the following sections.

2.1 Specifying the Elements to be Checked

Use the **Checker** form's **Add** and **Remove** menu options to edit the **Check Items** list until it shows those elements that you want to check.

The **Add** menu options work as follows:

- **CE**
  Adds the current element (as shown in the **Explorer** or **Members** List).
- **CE Members**
  Adds all members of the current element, but not the current element itself.
- **Pick**
  Lets you add elements by picking them in a graphical view, using any of the standard event-driven graphics options.
- **Failed List** (valid only after a previous data check)
  Adds all elements which have failed an earlier check, as listed on a **Checker Results** form (see *Running the Checks and Viewing the Results*). This option is useful for rechecking items after you have corrected the reasons for their earlier failure.
- **List** *list_name*
  Adds all elements in the named list (as created using the application's **Utilities > Lists** menu option). This provides a convenient way of adding elements which conform to a selection rule (for example, 'all pipes with pspec eq /A1A'), or which are within a given volume.

The **Remove** menu options work as follows:

- **All**
  Empties the **Check Items** list.
- **Selected**
  Removes all items currently selected in the **Check Items** list.
  (Pick an element in the list to select it; pick it again to deselect it.)
- **CE**
  Removes the current element (as shown in the **Explorer** or **Members** List).
- **CE Members**
  Removes all members of the current element, but not the current element itself.
- **Pick**
  Lets you remove elements by picking them in a graphical view, using any of the standard event-driven graphics options.
- **List** *list_name*
  Removes all elements in the named list (as created using the application's **Utilities > Lists** menu option). This provides a convenient way of removing elements which conform to a selection rule (for example, 'all pipes with pspec eq /A1A'), or which are within a given volume.

2.2 Selecting the Check Functions to be Carried Out

The available check functions are organised into **class** and **group** categories, simplifying the selection of those most relevant to your current design activities. Each class of checks represents, typically, a main design discipline; for example, Steelwork, Piping, HVAC, Cable Trays, etc. Each class may have one or more groups associated with it, representing more specific features of the design discipline; for example, HVAC Branches, HVAC Offsets, etc.

First select the required class of checks from the **Classes** options. This will automatically update the **Groups** options.

Now select the required group of checks from the **Groups** options. This will automatically update the **Checks** list to show all individual checks applicable to the chosen class and group.

From the **Checks** list, select one or more checks that are to be made on the elements in the **Check Items** list. (Pick an element in the list to select it; pick it again to deselect it.)

2.3 Highlighting Elements in the Graphical View

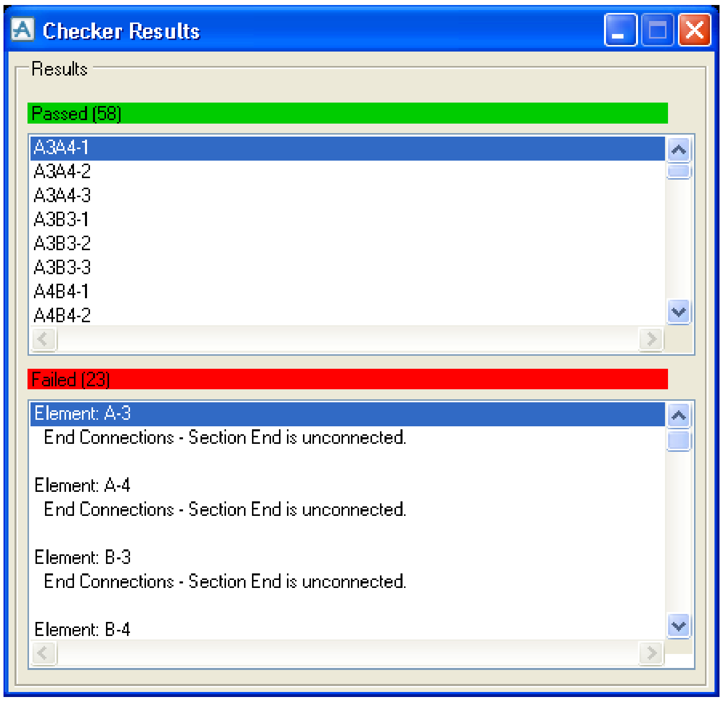
To make it easier to interpret the check results, you can highlight particular elements in the graphical view. The **Highlight** menu options work as follows:

- **Elements**
  Highlights all elements that are in the **Check Items** list. When you select this option, you will see a form that lets you choose the highlight colour. Note that only elements already in the **Drawlist** will be highlighted; elements are not added to the view automatically.
- **Passed**
  This is an On/Off toggle option. When on (shown by a tick on the menu), all elements which pass subsequent checks will be highlighted. Set the required highlight colour using the **Colours > Passed** menu option.
- **Failed**
  This is an On/Off toggle option. When on (shown by a tick on the menu), all elements which fail subsequent checks will be highlighted. Set the required highlight colour using the **Colours > Failed** menu option.
- **Clear**
  Clears either just the current element (**CE**), or everything in the graphical view (**All**).

2.4 Running the Checks and Viewing the Results

When you have set up the check requirements as described in *Specifying the Elements to be Checked*, *Selecting the Check Functions to be Carried Out* and *Highlighting Elements in the Graphical View*, click the **Check** button to carry out the checking operations. The results will be displayed in a **Checker Results** window, as shown.

:



A3A4-1
A3A4-2
A3A4-3
A3B3-1
A3B3-2
A3B3-3
A4B4-1
A4B4-2

Element: A-3
End Connections - Section End is unconnected.

Element: A-4
End Connections - Section End is unconnected.

Element: B-3
End Connections - Section End is unconnected.

Element: B-4

The **Passed** list shows all elements that have passed all specified checks. The **Failed** list shows all elements that have failed one or more checks, with a brief description of the reason for the failure.

2.5 Navigating to a Checked Element

To navigate easily to any element shown in the **Passed** or **Failed** list, click on that item in the list.

This facility is particularly useful if you want to navigate to a failed element to correct the reason for the failure before repeating the checks.

2.6 Saving Check Results to a File

To save the results of a data check to a file, select one of the following:

- **Control > Save > Results**

  This saves both the Passed and Failed results. Use the displayed **File Browser** to specify the required file name and its location.
- **Control > Save > Failed Reference**

  This saves the database references of elements that have failed, together with the checks performed. Use the displayed **File Browser** to specify the required file name and its location. This facility makes it easy to recheck the failed elements later.

To reload a list of failed element references (saved as in the preceding paragraph), typically following corrections to their data settings, select **Control > Load > Failed References** and use the displayed **File Browser** to specify the required file.

2.7 Updating the Available Check Functions

If the list of available checks has been modified during your current Outfitting session (say by editing existing checks or adding new ones), you can update the **Checker** form to show the new options immediately. To do so, select **Control > Reload Checks**.

The checks will be reloaded from the relevant default and user directories and the **Classes**, **Groups** and **Checks** options will be updated as appropriate.

3 Adding a New Check Function

3.1 Checker Configuration Files

Although the Checker utility is used by AVEVA to supply standard data checking facilities in Outfitting, its main purpose is to allow users to add their own checking tools.

The utility is controlled by creating **configuration files.** These configuration files will tell the system what checks are available, and which elements to select for checking. In order to keep standard (AVEVA) and user checking functions separate, they are configured in separate configuration files.

The standard AVEVA-supplied checks for Outfitting data are configured in a file called **com-checks.pmldat**, in the product directories. This file must not be modified. (The com prefix shows that this file can, in principle, apply to any Outfitting module; in this release, the utility is available from Outfitting Design only.)

User-supplied checking tools for Design data must be configured in files named **des-checks.pmldat** in the following directories:

- The `PDMSDFLTS` directory holds company-wide checks, which are available to all projects.
- The `ABCDFLTS` directory (where `ABC` is the project name) holds checks specific to one project.
- The `PDMSUSER` directory holds checks specific to a single user.

Entries in all des-checks.pmldat files available to a user will be combined together on the **Checker** form.

The format of des-checks.pmldat files is described by example below. The commands in the file must be valid PML syntax, and all information must be provided for each checking function.

The Checker file in the following example contains two checks:

- The first check identifies any panels that have incorrectly defined boundary curves. It does so by checking for panels having fewer than three vertices, or zero-length edges.
- The second check looks for structural elements that do not obey a naming convention. The naming convention for this example is that the names of all `SCTN`, `PANEL`, `FRMW` and `SBFR` elements must start with the first two letters of a `UDA :PRODNO`, which is set at `ZONE` level.

```
-- Data file for Design Checker Utility
```

`--` Comment lines begin --

```
--Checks for panel zero length edges, backtracks, and <3 vertices
```

`!Check = object CHECK()`	Declaration of new check entry
`!Check.Name     = 'LOOPCHECK'`	Name of check function
`!Check.Title    = 'Check Panel Boundary'`	Title appears in Check selection box
`!Check.Class    = 'Steelwork'`	Add this check function to this class
`!Check.Group    = 'Panels'`	Add this check function to this group
`!Check.Types    = 'PANE'`	Type of element to be checked
`!Check.Rule     = ''`	Rule for filtering selected element types (none)
`!Check.Function = '!!EdgeCheck'`	Name of Check Function
`!Check.Module   = 'Design'`	Outfitting module using this function
`!Check.FileType = '$1'`	Mandatory line for checker utility
`!!AddCheckerCheck(!Check)`	Adds this check to the list of checks

```
--

-- Check that primary steel element names begin with the first two
letters of the Production Number in :PRODNO UDA on ZONE
```

`!Check = object CHECK()`	Declaration of new check entry
`!Check.Name     = 'STRUCTNAME'`	Check name
`!Check.Title   = 'Check Steelwork Names’`	Title appears in Check selection box
`!Check.Class    = 'Steelwork'`	Class
`!Check.Group    = 'Administration'`	Group
`!Check.Types    = 'PANE SCTN FRMW SBFR'`	List of element types to check
`!Check.Rule     = 'FUNC OF ZONE eq |PS|'`	Selection filter
`!Check.Function = '!!PSNameCheck'`	Name of Check Function
`!Check.Module   = 'Design'`	Used in Outfitting Design
`!Check.FileType = '$1'`	Mandatory Line
`!!AddCheckerCheck(!Check)`	Add to the list of Checks

!Check.Title identifies the check in the **Checks** selection window and in the **Failed** results list.

!Check.Class and **!Check.Group** determine the **Classes** and **Groups** descriptions for the check function.

!Check.Types is a list of element types that will be selected for this test. This list will be filtered by the rule in **!Check.Rule**. Only elements from the !Check.Types list which obey any conditions specified in !Check.Rule will be presented to the check function by the Checker utility. The database reference (DBREF object) of a selected database element is passed to the check function. The check function must be written to handle any elements that might be selected and passed to it. If the check function itself fails, this will be reported in the **Failed** results list.

!Check.Function is set to the name of the check function, which must include the **!!** at the beginning of the name. This function must exist in the PMLLIB search path. The filename of the function will be the same as the function name, but all in lower case with a .pml filename extension; for example, the !!PSNameCheck function will be defined in file psnamecheck.pml.

!Check.Module is set to the Outfitting module in which this check will run.

The **!Check.FileType = '$1'** line is mandatory.

The **!!AddCheckerCheck(!Check)** line adds the details of this check to the list of available checks.

3.2 Rules for Organising Checks into Groups

Any number of Check Classes and Check Groups can be defined, but you must adhere to the following rule:

- All checks in the same class and group must have exactly the same selection criteria. !Check.Types must contain the same element list, !Check.Rule must contain the same selection filter, and !Check.Module must identify the same module.

If this rule is broken, then the results from the checker can be unreliable, particularly when multiple checks are run at the same time.

3.3 Check Functions

Check functions are PML functions which are called by the Checker utility. The name of the function is provided to the Checker utility from the !Check.Function line in a des-checks.pmldat file.

A Check function contains PML code, which must follow these rules:

- The function definition must have the following format:

  ```
  define function !!FunctionName ( !ItemRef is DBREF, !Check is
  CHECKDEFINITION ) is CHKRETURN
  ```

  !!FunctionName is the name of the function in the !Check.Function line in a des-checks.pmldat file. The PML file for a check function will be in a file named functionname.pml.

  !ItemRef is the name of the variable containing the database reference of the item to be checked.

  !Check is passed as an argument to the function, although it is not often used. !Check contains all of the information provided in the des-checks.pmldat file for this check function.

  These arguments are strictly read only. You should not reset the values of these arguments in the function itself.

- The return information from a Check function must be put into a **ChkReturn** object. The detailed definition of this object is described later in this guide.

  A ChkReturn object must be built by the Check function to tell the Checker utility whether the check has passed or failed. It is advisable to initialise the return value at the start of the Check function:

`!Result`	`= object CHKRETURN()`	Declare the Return object

`!Result.Passed = true`	Initialise to test passed
`!Result.Messages.clear()`	Initialise the message list

- The remainder of the function performs the required check on the object passed in the first argument of the Check function. If any test in the function fails, the result must be set to indicate a failure. This is done by using the following method:

`!Result.Passed = false`	Indicates a test failure
`!Result.Messages.append(‘Text of error message’)`	Add a message to the list of fail messages

The function may stop and return a result after the first error is found, or it could go on and find other errors and add them to the message list.

The result is returned to the Checker utility using the command:

```
return !Result
```

3.4 Checker Examples

3.4.1 Example 1: Panel Boundary Check

This example checks a panel boundary for zero length edges and missing vertices. The comments in the PML code describe the operations being performed.

```
------------------------------------------------------------------------
-- Description:
-- Checks for bad definition of panel boundary vertices:
-- Zero length edge; Less than three vertices; Panel loop not found
------------------------------------------------------------------------

define function !!EdgeCheck(!PanelRef is DBREF, !Check is CHKDEFINITION ) is
CHKRETURN

  -- Initialise Variables
  !Result        = object CHKRETURN()
  !Result.Passed = true
  !Result.Messages.clear()

  -- Get panel loop element - error if it does not exist
  !PloopRef = ( PLOOP 1 OF $!PanelRef )
  Handle any
    !Result.Passed = false
    !Result.Messages.append('No Panel Loop Found')
    return !Result
  endhandle

  -- get array of vertices belonging to the panel boundary
  VAR !Vertices COLLECT ALL PAVE FOR $!PloopRef
  !NumberOfVerts = !Vertices.size()

  -- Check that there are at least three vertices
  if(!Vertices.size() lt 3) then
    !Result.Passed      = false
    !Result.Messages.append('Only ' & !Vertices.size() & ' Vertices')
    return !Result
  endif

  -- Add first vertex to the end to close the loop
  !Vertices.append(!Vertices[1])

  -- Loop through vertices and check for zero length edge
  do !IndexVerts to !NumberOfVerts

    -- get address and position of this vertex and next vertex
     !Vertex        = !Vertices[!IndexVerts].dbref()
     !NextVertex    = !Vertices[!IndexVerts + 1].dbref()
     !VertexPos     = !Vertex.Pos
     !NextVertexPos = !NextVertex.Pos

     -- calculate distance between vertices & test for less than 0.01mm tolerance
     VAR !Dist CONSTRUCT DIST $!VertexPos TO $!NextVertexPos
     if(!Dist.real() lt 0.01mm) then
       !Result.Passed = false
       !Result.Messages.append('Zero length edge: Vertex ' & !IndexVerts.string())
       skip
     endif

   enddo

  -- Return Data
  return !Result
endfunction
```

The following is a sample report generated by running the preceding panel boundary check:

```
Check Report File

  Created By : M.Barlow
        Date : 10 Oct 97

Checks Performed
   Class : Steelwork
   Group : Panels
           [1] Check Panel Boundary


Summary of Checks

           Elements passed all tests : 8
    Elements failed on or more tests : 4
                               Total : 12

Elements that have passed all checks : 8

    DEVTEST-C22001-P00001
    DEVTEST-C22001-P00002
    DEVTEST-C22001-P00003
    DEVTEST-C22001-P00004
    DEVTEST-C22001-P00005
    DEVTEST-C22001-P00006
    DEVTEST-C22001-P00007
    DEVTEST-C22001-P00008

Elements that have failed one or more checks: 4
```

```
    Element: PANEL 1 of FRMWORK /DEVTEST-C22001
      Check Panel Boundary - No Panel Loop Found

    Element: PANEL 2 of FRMWORK /DEVTEST-C22001
      Check Panel Boundary - Only 1 Vertices

    Element: PANEL 3 of FRMWORK /DEVTEST-C22001
      Check Panel Boundary - Zero length edge: Vertex 2

    Element: PANEL 8 of FRMWORK /DEVTEST-C22001
      Check Panel Boundary - No Panel Loop Found

End of Check Report File
```

3.4.2 Example 2: Structural Element Name Check

This example checks the names of PANE, SCTN, FRMW and SBFR elements for primary steel ZONES (with FUNC ‘PS’). This example checks a panel boundary for zero length edges and missing vertices. The comments in the PML code describe the operations being performed.

```
------------------------------------------------------------------------
-- Description:
-- Checks that the first two letters of primary steel element names are
-- the same as the first two letters of the Production Number of that
-- ZONE. The Production Number is stored in UDA :PRODNO. A Zone containing
-- primary steel has its FUNC attribute set to 'PS'. Errors tested:
-- Production Number UDA not set
-- Production Number UDA incorrect (less than two letters)
-- Steelwork element not named
-- Name of <item full name> does not begin with <first 2 letters of :PRODNO>
------------------------------------------------------------------------

define function !!PSNameCheck (!ItemRef is DBREF, !Check is CHKDEFINITION ) is
CHKRETURN

  -- Initialise Variables
  !Result        = object CHKRETURN()
  !Result.Passed = true
  !Result.Messages.clear()

  -- Get Production Number and check that it is set
  !ZoneRef = ( ZONE OF $!ItemRef )
  !ProductionNumber = !ZoneRef.attribute(':PRODNO')

  if ( !ProductionNumber.empty() ) then
    !Result.Passed = false
    !Result.Messages.append('Production Number not set for ' & !ZoneRef.flnn)
    return !Result
  endif

  -- Check that production number has at least two characters
  if ( !ProductionNumber.length() lt 2 ) then
    !Result.Passed      = false
    !Result.Messages.append('Production Number incorrect for ' & !ZoneRef.flnn)
    return !Result
  endif

  -- Get first two characters of the production number
  !ProdCode = !ProductionNumber.substring(1,2)
  -- Get name of steel element
  !Name = !ItemRef.Name

  -- Test for unset name - First character will be '='
  if(!Name.substring(1,1) eq '=') then
    !Result.Passed = false
    !Result.Messages.append(!ItemRef.Type & ' not named ')
    return !Result
  endif

  -- Test for first two letters of Production number = first two letters of name
  if( !Name.substring(2,2) neq !ProdCode ) then
    !Result.Passed = false
   !Result.Messages.append( !ItemRef.Type & ' name does not begin with ' & !ProdCode
)
  endif

  -- Return Data
  return !Result
endfunction
```

The following is a sample report generated by running the preceding panel name check:

```
Check Report File

  Created By : M.Barlow
        Date : 9  Oct 97

Checks Performed
   Class : Steelwork
   Group : Administration
           [1] Check Primary Steelwork Names

Summary of Checks

           Elements passed all tests : 13
   Elements failed one or more tests : 5
                               Total : 18

Elements that have passed all checks : 13

    PA-C22001-P00002
    PA-C22001-S00001
    PA-C22001-P00003
    PA-C22001-S00003
    PA-C22001-S00004
    PA-C22001-S00005
    PA-C22001-S00006
    PA-C22001-S00007
    PA-C22001-S00008
    PA-C22001-S00009
    PA-C22001-P00005
    PA-C22001-P00006
    PA-C22001-P00007

Elements that have failed one or more checks: 5
```

```
    Element: DEVTEST-C22001
      Check Primary Steelwork Names - FRMW name does not begin with PA

    Element: DEVTEST-C22001-P00001
      Check Primary Steelwork Names - PANE name does not begin with PA

    Element: DEVTEST-C22001-S00002
      Check Primary Steelwork Names - SCTN name does not begin with PA

    Element: DEVTEST-C22001-P00004
      Check Primary Steelwork Names - PANE name does not begin with PA

    Element: PANEL 5 of FRMWORK /DEVTEST-C22001
      Check Primary Steelwork Names - PANE not named

End of Check Report File
```

4 Modifying and Deleting Check Functions

If any of the information in the des-checks.pmldat configuration file is changed, or if a check is deleted by removing its entry from the file, it will be necessary for the checks to be reloaded on each active or saved user Checker utility form by using the **Control > Reload Checks** option.

The Check functions themselves can be modified by simply editing the Check function PML file. If the Check function is moved to another location, it will not be found until users restart Outfitting, or the **PML REHASH** command is used to tell Outfitting to rebuild its table of currently available PML functions.

5 Checker Objects

A ChkReturn object is returned from a Check function to the Checker utility to tell the utility whether the element has passed or failed the test.

If the check has passed, the **.Passed** member will be set to **TRUE** and the array of text string messages is initialised as empty.

If the check has failed, the **.Passed** member will be set to **FALSE** and the array of text string messages should contain at least one entry.

```
-- Define object
define object CHKRETURN

  -- Message Array
  member .Messages is ARRAY
  -- Passed=TRUE   Failed=False
  member .Passed   is BOOLEAN

endobject
-- End of object definition
```

The **Check** object is the object created in the des-checks.pmldat file to configure the Checker utility. It is also passed to the Check function as its second argument.

```
-- Define object
define object CHECK

  -- Check Name (unique)
  member .Name     is STRING
  -- Check Class
  member .Class    is STRING
  -- Class Check Type
  member .Group    is STRING
  -- Check Description
  member .Title    is STRING
  -- Check Function Name
  member .Function is STRING
  -- Permissible Element types
  member .Types    is STRING
  -- Selection rule
  member .Rule     is STRING
  -- Successful action
  member .Passed   is STRING
  -- Failed action (if none returned from check)
  member .Failed   is STRING
  -- Module check is available form
  member .Module   is STRING
  -- File type (who/where check is loaded from)
  member .FileType is STRING

endobject
-- End
```

Index