Undocumented Features Only

1999, $UUU press

Uniface will not support undocumented features. If you find its not working quit right don't use it. Undocumented features can be removed from future Uniface versions or may behave different from version to version. However some of the features are used inside the forms of the IDF/UDE and are likely to be maintained as long as the IDF/UDE is using it.

$print_assignments

Name	`$print_assignments` **Prints the assignments used, for this session, in the transcript window**
Synopsis	`$print_assignments [true|false]`
Description	`$print_assignments` **in the** `SETTINGS` **section of the** `asn` **file prints the assignments defined in the asn files to the transcript window. Only the assignments written after this setting are printed.** **Tested in 7.2.03**
Example	`$print_assignments`

$trace_is_true

Name	`$trace_is_true` **Enables the debug everywhere pressing the SWITCH KEY (<GOLD>Y)**
Synopsis	`$trace_is_true [true|false]`
Description	`$trace_is_true` **in the** `SETTINGS` **section of the** `asn` **file enables the debug window pressing the SWITCH KEY everywhere, without use the** `debug` **statement.** **Tested in 7.2.01. 7.2.02, 7.2.03**
Example	`$trace_is_true`

askmess

The "official" features are described in the proc manual. This document contains only undocumented features

Name	`askmess` **Display a message and wait for the user’s response.**

Synopsis	`askmess{/nobeep}{/question|/info|/warning|/error} MessageText {, Replies}`
	The message is of the form
	`Message{~Title}`
	The Replies is of the form:
	`Reply 1 , Reply 2 , ..., Reply n`

Use	**Allowed in form components.**
Description	**You can specify a title for the askmess window writting after the message a tilde (~) and the title string** **Tested in 7.2.03** **Documented by virginia-it-services**
Example	`askmess "Are you sure you want to quit? ~Exit confirmation"`

cont_field

Name	`cont_field`
Synopsis	`cont_field` *`FieldName`*
Use	Allowed only in form components Tested in 7.2.03
Description	`cont_field` **Breaks the current proc entry or trigger, positioning the cursor in the specified field**
Example	`cont_field "FIELD_1"`

display_length

Name	`display_length` **Returns the number of bytes of a field**
Synopsis	`display_length` *`FieldName`*
Use	**Allowed in all components.**
Description	`display_length` returns the number of bytes Uniface will attempt to store in the database. When dealing with multi-byte character sets (like Japanese) its a lifesaver because `length` returns the number of bytes Uniface will display on the screen. **<u>Supported since 7.2.03</u>**
Return Value	In `$result` the number of bytes

Example	`display_length field1`
	`message "The field length %%display_lenth "`

getvalrep

Name	`getvalrep`
Synopsis	`getvalrep` *`field, $register`*
Use	Allowed in form components
Description	`getvalrep` **retrieves the valrep list of a field. No differences detected with** `$valrep(field)` **excepts that only works with registers** Tested in 7.2.03

Example	`$LIST$ = "W=Wait;P=In process;E=End"`
	`setvalrep p_status, $LIST$`
	`getvalrep p_status, $1` ; $1 = $LIST$

strip_attributes

Name `strip_attributes`

Removes all the attributes and garbage in a field

Synopsis `strip_attributes` *`source, target`*

Use **Allowed in form components, services and reports**

Description **Copy the** `source` **field in** `target` **removing all attributes (bold, italics and underline), control characters, frames and rulers.**

Tested in 7.2.03

Return Value `$status` **contains**

$status	**Stripped**
0	**Nothing**
1	**Attributes**
2	**Garbage**
4	**Frame**
8	**Rule**

When more than one type of chars are removed, $status contains the sum of this values.

Example

```
strip_attributes a, b
if ($status > 0 )
   message "Field stripped...."
endif
```

$about

Name $about
Return a list of the current installation parameters

Synopsis $about

Use **Allowed in form components (and in service and report components that are not self-contained).**

Description **The function** $about **returns a list of the current installation parameters.**

Tested in 7.2.02, 7.2.03

Note: Refer to Proc Language Manual section 1.12.3 Lists of items for information about the structure of lists.

Return Value **The function** $about **returns a string that contains a list with this keys:**

key	Description
version	The Uniface Release in the "About UNIFACE" dialog box (Ex. 7.2)
track	The Update of the release (Ex. 7.2.03)
config	The service-pack Update of the release (Ex. u-sts302.b23)
date	Date of the last version update (Ex. November 23,1998)
platform	Platform code (Ex. MS1)
update	?
tag	?
GEN	?
LST	?
VER	?
FRM·!	?
NT	?

See the "About UNIFACE" in your IDF...

Example

```
getitem/id trk, $about, "track"
if (trk = "7.2.03")
   activate "workaroudn".exec()   ; Service to solve a bug
in 7.2.03
endif
```

condition

The "official" features are described in the proc manual. This document contains only undocumented features

Name condition

Return the result of evaluating a conditional expression.

Synopsis condition(*Expression[,list])*

Use **Allowed in form, service, and report components.**

Description **You can use a second (and optional) parameter a associative list (id=value) that** condition **should use to evaluate the expression as variables and values.**

Tested in 7.2.03

Example

```
amount = 1500
limits = "MIN=$number(1000);MAX=$number(2000)"        ;
limits of amout in a list
exp = "amount > MIN & amount < MAX" ; exp is a string with
the expression
if (condition(exp, limits))
   message "The amount is correct"
else
   message "Not between limits"
endif
```

Note **The evaluation of condition is done using (by default) string types and this can have some problems :**

```
limits = "MIN=1000;MAX=2000"
$1 = condition("20 < MIN",limits)
```

In this case $1 **is FALSE (0), because the condition evaluates the string "20" less than the string "1000" (ASCII values).**

In order to solve this you must assing the type to the value of the items as :

```
limits = "MIN=$number(1000);MAX=$number(2000)"
$1 = condition("20 < MIN",limits)
```

This works fine!!!

The $typed fuctions are :

- $string(*value*)
- $syntax(*value*)
- $number(*value*)
- $float(*value*)

- `$boolean(`*`value`*`)`
- `$date(`*`value`*`)`
- `$datim(`*`value`*`)`
- `$time(`*`value`*`)`
- `$clock(`*`value`*`)`

expression

The "official" features are described in the proc manual. This document contains only undocumented features

Name	expression **Return the result of evaluating a nonconditional expression.**
Synopsis	expression(*Expression[, list]*)
Use	**Allowed in form, service, and report components.**
Description	**You can use a second (and optional) parameter a associative list (id=value) that** expression **should use to evaluate the expression as variables and values.** **Tested in 7.2.03**

Example

```
amount = 1000
taxes = "VAT=6;SPECIAL=4" ; Note that taxes is a string
(variable or field) and
                          ; VAT and SPECIAL are id of two
items

exp1 = "amount + (amount * VAT /100 )" ;exp1 and exp2 are
string with the "formula"
v1 = expression ( exp1, taxes ) ; v1 is 1060

exp2 = "amount + (amount * VAT /100 ) + (amount * SPECIAL /
100)" ;
v2 = expression ( exp2, taxes ) ; v2 is 1100
```

idpart

Name	`idpart`
Synopsis	`idpart(`*`item`*`)`
Use	Allowed in form, service, and report components.
Description	`Idpart` returns the identification of a list `item`. All list can be processed as indexed and associative. If you get a item from an associated list but using an index you can obtain the identification using this function **Tested in 7.2.03**

Example	`mylist = "A=hello;B=world;C=uuu"`
	`getitem myitem, mylist,1   ; Read the first item of list (mylist is : "A=hello"`
	`mykey = Idpart(myitem)     ; mykey is : "A"`

item

Name	`item` get a item from a list
Synopsis	`item(`*`key, list`*`)`
Use	Allowed in form, service, and report components.
Description	`item` **returns the value of a item in the** `list` **wich matches with the specific** `key`**. This is the same functionality than** `getitem/id` **but as function** **Tested in 7.2.03**
Example	**Supose that** `mylist = "A=hello;B=world;C=uuu"` **then**

```
call my_entry(item("B",mylist)) ; item function used in a call statement
```

... this is equivalent to ...

```
getitem/id myvar, mylist, "B")
call my_entry(myvar)
```

$ustat

Name	`$ustat` **Return some values about devices tables**
Synopsis	`variable = $ustat(Value)`
Use	**Tested in form components.**
Description	**This function allows access to the dimension of a device table depending defined for any Device Mode (0-7) (Defined Print Job Model,...)** **This function can be useful in order to configure automatically the printers needed for your application.** **<u>Tested in 5.2.g and 7.2.03</u>**
Return Value	**The value pased to this function MUST the constant value, no variables, fields or registers allowed.**

Param	**call**	**Description**
`$result = <DEVICE_TYPE>`	`$1 = $ustat(19)`	**init for consult a device type. Return 0 when success**
`$result = 1`	`$1 = $ustat(30)`[1]	**Return the Horizontal size of mode 0**
`$result = 2`	`$1 = $ustat(30)`[1]	**Return the Vertical size of mode 0**
`$result = 3`	`$1 = $ustat(30)`[1]	**Return the Width of mode 0**
`$result = 4`	`$1 = $ustat(30)`[1]	**Return the Height of mode 0**
`$result = 0`	`$1 = $ustat(30)`[1]	**In `$result` the description of this mode**

[1]Use 31 trhu 37 for modes 1 thru 7, and the value 20 for the current display.

Example

```
entry describe_mode
params
   string device : IN
   numeric mode : IN
   string description : OUT
   numeric hsize : OUT
   numeric vsize : OUT
   numeric width : OUT
   numeric height : OUT
endparams
variables
   string dummy
endvariables

$result = device
$result = $ustat(19)
if ($result = 0)  ; if exists
   selectcase mode
   case 0         ; values for mode 0
      $result = 0
      dummy = $ustat(30)
      description = $result
      $result = 1
      hsize = $ustat(30)
      $result = 2
      vsize = $ustat(30)
      $result = 3
      width = $ustat(30)
      $result = 4
      height = $ustat(30)
    case 1         ; values for mode 1
      $result = 0
      dummy = $ustat(31)
      description = $result
      $result = 1
      hsize = $ustat(31)
      $result = 2
      vsize = $ustat(31)
      $result = 3
      width = $ustat(31)
      $result = 4
      height = $ustat(31)
    case ....
     ....
    endselectcase
endif
```

```
end
```

valuepart

Name	`valuepart`
Synopsis	`valuepart(`*`item`*`)`
Use	Allowed in form, service, and report components.
Description	`valuepart` returns the value of a list `item`. All list can be processed as indexed and associative. If you get a item from an associated list but using an index you can obtain the value using this function **Tested in 7.2.03**

Example	`mylist = "A=hello;B=world;C=uuu"`
	`getitem myitem, mylist,1   ; Read the first item of list (mylist is : "A=hello"`
	`mykey = valuepart(myitem)     ; mykey is : "hello"`

$workfilesize

Name	`$workfilesize` Amount of memory used
Synopsis	`$workfilesize`
Use	Allowed in form, service, and report components.
Description	`$workfilesize` returns the amount of memory used **Documented by virginia-it-services**
Example	`message " The amount of memory used is %%$workfilesize"`

/nodebug

Name	`/nodebug` **Sub-switch of compiling options**
Synopsis	`idf [/frm | /svc | /rpt ] {/nodebug}`
Description	Disables the debug mode for the compiled components **Available in 7.2.04**
Example	`idf /frm myform /nodebug`

$tometa

Name	`$tometa` **Return the character coresponding to an ASCII code**
Synopsis	`variable = $tometa(Code)`
Use	**Available in any component type.**
Description	**The character corresponding to the specified ASCII *`Code`* is assigned to *`variable`*.** **<u>Tested in 7.2.03 and 7.2.04</u>**

Example	`$1 = $tometa(36) ; $1 = "$"`
	`b = 85`
	`$2 = $tometa(b) ; $2 = "U"`

show

Name	`show`
Synopsis	`show`
Use	Allowed only in form components
Description	`show` **This new proc statement will display the current component and return to the proc code, so it is posible, for example, to have a Merter or a Slider widget displaying a percentage completed. Proc Statement.** **This new proc statement will display the current component, usefull meters and progress bars.** **Only in 7.2.04**

Example

```
meter_value = 100
setocc "entity",1
while ($status >= 0)
; perform procesing ...
meter_value = meter_value - 1
show
setocc ...
endwhile
```

setvalrep

Name	`setvalrep`
Synopsis	`setvalrep` *`field, List`*
Use	Allowed in form components
Description	`setvalrep` **assings a list to the valrep of a field. No differences detected with** `$valrep(field) = list` **Tested in 7.2.03**

Example	`$LIST$ = "W=Wait;P=In process;E=End"`
	`setvalrep process_status, $LIST$`

fieldcopy

Name	Fieldcopy **Copy the field source into a field trace.**
Synopsis	Fieldcopy(*Field_Source, Field_Target)*
Description	The Fieldcopy statement only copies the contents of the field specified in the source to the target. This undocumented statement provides no new understanding because is like an assignment. **Tested in 7.2.04**
Return Value	In $status = 0 if the operation finish successfully.
Example	Fieldcopy(field1.table1, dummy_field.dummy_table)

trace

Name `trace`

Writes trace information into the message frame

Synopsis `trace`

Use **Available in any component type.**

Description `trace` statement write information about driver operations done between two `trace` statements

<u>Tested in 7.2.03 and 7.2.04</u>

Example

```
trace               ; starts trace
;
; code to trace (retrieve, store, edit,....)

trace               ; end trace
```

The traced code can generate in your message frame an output like :

```
(Trace ) Elaps: 405; Cpu: 405; Ret: 0; Sto/Rem: 6; Cle/Rel:
1
-----> Inp: 0; Out: 6; Dbms: 86; Dis: 124; Edit: 0; Misc: 0
```

$traceprint

Name `$traceprint`
Traces the fired triggers into the message frame

Synopsis `$traceprint =` *`Value`*

Use **Available in any component type.**

Description When `$traceprint` is enabled a line is write in your message frame for every fired and end trigger or other conditions depending on the specified *`Value`*.
Every line describes the trigger, the exit `$status` and entity and occurence involved.

The values assigned can be :

1	Validation info
2	Fired trigers
4	Transient proc errors
8	Proc errors
16	Activate errors

If you wish trace more than one possibilities, just sum it (as `$ioprint`)

Available and tested in 7.2.04

Example

```
$traceprint = 1
;
; your code
;
$traceprint = 0
```

The results message frame can be :

```
End <CLR > es TRACE, status 0
Act <RETR> es TRACE
Act <READ> ent E1, occ 1
End <READ> ent E1, occ 1, status 0
End <RETR> es TRACE, status 0
End <EXEC> es TRACE, status 10
End <EXEC> es UUEXTN65, status 0, $prompt =
CHECK4.DUMMY.STANDARD
End <MNUS> es UUEXTN23, status 0
Act <OGF > ent UFORM, occ 1
End <OGF > ent UFORM, occ 1, status 0
```

$interactive

Name	`$interactive` **Allow to know is the user has in the interactive session.**
Synopsis	`$interactive`
Description	With $interactive you can check of the form is already in edit-mode or not. It returns zero, when the edit or display instruction was not executed (yet) **Tested in 7.2.04**
Return Value	$status = 0 the user doesn’t start an interactive session, before edit or display. $status > 0 (2) after the edit or display,
Example	`If (!$interactive) then proces_something`
Source	Hans Hoogerwerf h.hoogerwerf@cypres.nl

$componentinfo

Name	`$componentinfo` **Return information about the component.**
Synopsis	`$componentinfo(<componentname>,"VARIABLES")`
Description	The function $componentinfo with the switch VARIABLES return a list of component variables $VAR$. **Tested in 7.2.04**
Return Value	If $status = 0 return a list of component variables.
Example	`Slist = $componentinfo("%%$componentname","VARIABLES")`
Source	Hans Hoogerwerf h.hoogerwerf@cypres.nl

interrupt

Name	`interrupt` **Some interactions with the file-system**
Synopsis	`interrupt(`*`InterruptCode`*`,`*`ResultString`*`)`
Use	**Allowed in form components (and in service and report components that are not self-contained).**
Description	**The `interrupt` statement performs some actions under your file-system depending on the *InterruptCode* as show in the following table :**

InterruptCode	**Description**
0	Return file information
1	?
2	Return the dirname of a file
3	? (=2 ?)
4	?
5	Return the current directory
6	Concat the name of file with de current directory
7	returns the parent directory of the specified directory
8	?
9	?
10	Return a list of files in the current directory depending on the wildcard mask.
11	Return a list of subdirectories in the current directory
12	?

Tested in 7.2.02, 7.2.03

Note: Refer to Proc Language Manual section 1.12.3 Lists of items for information about the structure of lists.

Return Value

InterruptCode	***ResultString***	**Result description**
0	FileName	If the file exists : $result = <InformationFileList> $status = 0 If the file doesn't exists: $status = -1
2	FileName	$result contains the basename of the FileName (cut the dirname). Ex. if FileName is "/usr2/pepe.txt" $result contains "pepe.txt"
5		$result contains the name of the current directory
6	FileName	$result contains the concat of previous $result with FileName as needed for your file-system.
10	Mask	$result contains a list of the files which matches with the Mask (The mask can be a path + wildcard Ex. "/usr2/*.txt") $status contains the number of files in this list NOTE: if you call interrupt(10) $result contains a list of all the files in the current directory and $status the count
11	Mask	$result contains a list of the directories which matches with the Mask (The mask can be a path + wildcard Ex. "/usr2/dir*") $status contains the number of directories in this list NOTE: if you call interrupt(11) $result contains a list of all the subdirectories in the current directory and $status the count

The InformationFileList is a associative list with the keys `Path, DiskDir, Name, Type, Version, Attrib, RecSize` **and** `Filesize`

Example

```
file_name = "pepe.txt"
interrupt(0,file_name)
if ($status = -1)
   interrupt(5)
   message/error "File not found in directory %%$result"
else
   getitem/id fsize, $result, "Filesize"
   message "The file size of %%file_name is %%fsize"
endif
```

Some $trc_ settings

There exists a few $trc setting which offer you trace information into a log file.

Setting	Descripton and values
`$trc_start`	The name of the log file
`$trc_enable`	
`$trc_info`	The kind of information inclosed in the log file. Can be set to one or many of this keywords: `none, all, num, pid, prg, cat, lvl, dtm,nl`
`$trc_type`	Values allowed : `startup, cached, direct`
`$trc_levels`	A value in hex (9E seems to work)
`$trc_frame`	

Source Alex Leguevaques

A good combination which can be use to debug strange errors is :

```
$trc_type=direct
$trc_levels=9E
$trc_info=all
$trc_start=sys$login:uni_trace.log
```

This generate a file which looks like this :

```
 1     1e      932645065 ( 16619 if_7204]supra.exe;16) RDB_I_DBDRV  IO
Path=0 entity=none
 1     2e      932645065 ( 16619 if_7204]supra.exe;16) RDB_I_SELFID IO
Drv. U4.1
 1     3e      932645065 ( 16619 if_7204]supra.exe;16) RDB_I_DRVPAR IO
Drv.param O, D 5,E,S 200,L,R,Y
 1     4e      932645065 ( 16619 if_7204]supra.exe;16) RDB_I_DBRTN  IO
Drv. returns 0 in udrverr
 1     5e      932645065 ( 16619 if_7204]supra.exe;16) RDB_I_EDBDRV LO
Exit=0
 1     6e      932645065 ( 16619 if_7204]supra.exe;16) RDB_I_DBDRV  LO
Path=0 entity=none
 1     1e      932645065 ( 16619 if_7204]supra.exe;16) RDB_I_SQL    LO
Starting
 exec. imm. of CONNECT TO 'ATTACH ALIAS UNIF7$BASE FILENAME UNIF7$BASE
DBKEY SCOPE IS ATTACH' AS 'UNIF7$BASE'
```

/ drv

Name `/drv`

Generates an executable file for DB driver testing

Synopsis `idf /drv` *`DriverCode`*

Description Generates an executable file (`drvtst`) which you can run. This program starts a few tests (SELECT, INSERT, UPDATE, DELETE) with a time.

The `drvtst` program admits this parameters from command line:

```
/ASN=asn_file  (asn file)
/PRI=nn        (I/O message code)
/LOG=?|?|?     (Dbms/Network logon
information)
POLY           (To indicate Network and/or
record conversions)
FILes=n        (Number of files/tables to
process [1-8])
MIN=nnnn       (Minimum size of variable
field)
MAX=nnnn       (Maximum size of variable
field)
```

Tested in 7.2.03 (Solaris) and 7.2.04 (OpenVMS)

Example

```
idf /drv INF            # drvtst generation

drvtst /asn=idf.asn /pri=255   # Now test the driver using
drvtst...
```

Source Alex Leguevaques

icomp

The icomp perform enables you to use some uniface function available only from the IDF environment (at this moment ...). Using this functionality you can write your own installation programs in Uniface.

Click [here](#) for download a sample program of the icomp functions.

USE : idf /tst icomp

The normal use of the icomp is :

- The parameters are passed in the $9x global registers

- The function to perform is codified in $99

- Perform "icomp"

- $status < 0 if any error occurs

Functions described in this document

Function table

Compile components. Function 0		
Register	***Description***	***Values***
$99	**Function code**	**0**
$98	**Component name**	

$97	**Component type**	**0 (form) 1 (service) 2 (report)**
$96	**Skeleton**	**0 (no) 1 (yes)**
$95	**Proc listing**	**0 (no) 1 (yes)**
$94	**Info**	**0 (no) 1 (yes)**

Generating a zip component requires previous initializations....

Clear component. Function 3		
Register	***Description***	***Values***
$99	**Function code**	**3**

Compile objects to UOBJECT. Function 20		
Register	***Description***	***Values***
$99	**Function code**	**20**
$98	Object name	
$97	Library	
$96	Language	

Analyze model. Function 99		
Register	***Description***	***Values***
$99	**Function code**	**99**
$98	**Model name**	
$94	**Fixed value**	**0**

Check if a driver is installed. Function 110		
Register	***Description***	***Values***

$99	**Function code**	**110**
$98	**Driver code**	eg. INF, SYN, UWE, ORA...
When the driver is installed the value returned in `$status` is 0 (zero)		

Get information about the current DOL file. Function 189		
Register	***Description***	***Values***
$99	**Function code**	**189**

After perform this function the `$result` register will contains a list with two items :
`CREATE=<creation_date>;NAME=<dol_file_name>`
`Eg. CREATE=19990126000000;NAME=uobj.dol`		

Update the DOL file. Function 190		
Register	***Description***	***Values***
$99	**Function code**	**190**

Import objects from a trx file. Function 191		
Register	***Description***	***Values***
$99	**Function code**	**191**
$98	**Source file name**	
$97	**Fixed value**	**1**

Export object in a trx file. Function 192 (common parameters)		
Register	***Description***	***Values***
$99	**Function code**	**192**
$98	**Target File name**	
$97	**Supersede**	**0 (no) 1 (yes)**

Depending of the object you will export the registers you will need are:

Object	$92	$93	$94	$95	$96
Component [1]				Component name	16
Start Up shell				Application name	18
Model				Model name	38
Library				Library name	58
Global registers			""	Library name	50
Global procs	"P"	Proc name	Library name	"P"	26
Includes	"P"	Include name[2]	Library name	"I"	26
Messages	Language	Text name	Library name	"M"	26
Panels	"C"	Panel name	Library name	"C"	26
Devices T.T.	"T"	Device name	Library name	"D"	26
Keyboard T.T.	"T"	Keyboard name	Library name	"T"	26
Menus	Language	Menu name	Library name		88
Glyph	Language	Glyph name	Library name	"I"	92
Entity I. Template			EIT name		42
Field I. Template			FIT name		44
Field S. Template			FST name		46
Field L. Template			FLT name		48
Field Template			FT name		80

Signatures			**Signature name**	**"SYSTEM_LIBRARY"**	**100**
Constants	**"P"**	**"DEFPARAM"**	**Library name**	**"I"**	**26**

[1]**To complete the component export operation you should do after the icomp :**

```
$96 = 26
$94 = <componentname>
$95 = "I"
93 = "!DEFPARAM" ; ! is <gold>!
$92 = "P"
perform "icomp"
```

[2] Specify &!DEFPARAMS in the include name profile (&! Are <gold>&<gold>!

Export data. Function 300		
Register	***Description***	***Values***
$99	**Function code**	**300**
$98	**Target in the format DRV:ENTITY.MODEL**	
$97	**Source in the format DRV:ENTITY.MODEL**	**1 (form) 2 (service) 3 (report)**
$96	**Supersede**	**0 (yes) 3 (no)**
$94	Map file	
$93	Translation table	

Create table (SQL Script). Function 400		
Register	***Description***	***Values***
$99	**Function code**	**400**
$98	**Table in format DRV:ENTITY.MODEL**	
$96	**Fixed valued**	**0**

$97	**The result script is returned in this register**	

Dectect idf mode (Development/Deployment). Function 999		
Register	***Description***	***Values***
$99	**Function code**	**999**
If the idf mode is deployment the value returned in `$99` is `998`, in other case is `999`		

/dbg

Name	`/dgb` **Starts the idf environment in debug mode**
Synopsis	`idf /dbg`
Description	Starts the debug at the beginning of the <APPLICATION EXECUTE> trigger of the idf application. Notice that the first statement is `nodebug,` **Tested in 7.2.03**
Example	`idf /dbg`