Debugger_Fetch Function

Description

This function provides methods for accessing variable data within a Debugger Intercept routine.

Syntax

variableData = **Debugger_Fetch**(*method*, [*arg1, arg2*])

Parameters

The function has the following parameters:

Parameter	Description
method	The Debugger_Fetch function accepts various methods to access variables and their contents when OpenInsight has a Debugger Intercept routine in use. ***Note: The Debugger Intercept is set in the Environment Settings. To access the Environment Settings, select Environemnt Setting menu option from the Database menu in the Database Manager. In the Debugger Settings group, Intercept should be selected in the radio options and a stored procedure should be selected as an Intercept Routine. The Intercept Routine defines which stored procedure should execute when an unhandled error condition is encountered.***
arg1	Dependent upon the method.
arg2	Dependent upon the method.
arg3	Dependent upon the method.

Returns

An @fm-delimited list of data. The data returned is dependent upon the method.

Remarks

The Methods and their return values are only accessible when in Debugger Intercept mode.

See also

Debugger Intercept functionality.

DEBUGGER_DUMP is a SYSPROG subroutine containing sample code for using the Debugger_Fetch() routine. The source code is published below.

Method/Argument/Return Value Chart

Method	arg1	arg2	arg3	Return Value	Note
LISTLOCAL	n/a	n/a	n/a	An @fm-delimited list of @vm-delimited variable names and types and subcripts.	If the variable is a dimensioned array, the subscripts are returned as an @svm-delimited array.
LISTLABELLED	n/a	n/a	n/a	An @fm-delimited list of @vm-delimited Labelled Commons, variable names, types and subscripts.	If the variable is a dimensioned array, the subscripts are returned as an @svm-delimited array.
LISTCOMMON	n/a	n/a	n/a	An @fm-delimited list of @vm-delimited Labelled Commons, variable names, types and subscripts.	This type is not seen in OpenInsight and is mentioned due to AREV32.
GETLOCAL	Variable Name or Position	[Row Number]	[Column Number]	The value of the variable.	If the variable is a Dimensioned Array, Row Number and Column Number wil retrieve the data from specific positions within in the array.
GETLABELLED	Label Name	Variable Name or Position	n/a	The value of the variable within the COMMON.	
GETCOMMON	Common Name	Variable Name or Position	n/a	The value of the variable within the COMMON.	

GETLOCALDIMTYPE	Dimensioned Array Name	Row Number	Column Number	The type of the variable within the COMMON.	
GETLABELLEDDIMTYPE	Dimensioned Array Name	Row Number	Column Number	The type of the variable within the COMMON.	
GETCOMMONDIMTYPE	Dimensioned Array Name	Row Number	Column Number	The type of the variable within the COMMON.	

Sample Code

```
compile subroutine Debugger_Dump( void )

/*
   Author   : Meester C
   Date     : 21 October 2008
   Purpose  : Simple function to grab all the variable contents in a broken
              proc and write them out to a "DEBUGGER_DUMP" record in SYSLISTS

   Comments
   ========

   This is a very basic procedure. It would really be a good idea to monitor
   the size of the dump output and break it up into sections and write it out
   as an OS file for example.  The output has the potential to be extremely
   large due to the possibility of large variable contents and iterating
   through dimensioned arrays with large values.

   ... but that is another story ...


   Amended  Version  Date         Reason
   =======  =======  ====         ======
*/
   declare function debugger_Fetch
   $insert debugger_Fetch_Equates
   $insert logical

   equ VERSION$       to "1.0.0"
   equ LOCAL_DUMP_ID$ to "oe_local_dmp.txt"
   equ LCOMM_DUMP_ID$ to "oe_lcomm_dmp.txt"
   equ CRLF$          to \0D0A\
   equ CR$            to \0D\
   equ LF$            to \0A\
   equ TAB$           to \09\

   goSub dumpLocalVars
   goSub dumpLabelledCommonVars

return
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

dumpLocalVars:
   localList  = debugger_Fetch( "LISTLOCAL" )
   localVars = ""

   xCount = count( localList, @fm ) + ( localList # "" )
   for x = 1 to xCount
      var = localList<x>
      varName = var[1,@vm]
      varType = var[col2()+1,@vm]
      varDim  = var[col2()+1,@vm]

      begin case
         case ( varType = VAR_TYPE_DESC_UNASSIGNED$ )
            localVars := varName : @vm : varType : @vm : "" : @fm
```

```
         case ( varType = VAR_TYPE_DESC_IDISPATCH$ )
            localVars := varName : @vm : varType : @vm : "" : @fm

         case ( varType = VAR_TYPE_DESC_DIM_ARRAY$ )
            localVars := varName : @vm : varType : @vm : "" : @fm

            * // We need to iterate across the matrix ...
            dim1 = varDim[1,@svm]
            dim2 = varDim[col2()+1,@svm]

            d1 = 0
            d2 = 0
            if len( dim2 ) then
               dVar = debugger_Fetch( "GETLOCALDIMTYPE", varName, d1, d2 )
               goSub processLocalDimVar
            end else
               dVar = debugger_Fetch( "GETLOCALDIMTYPE", varName, d1 )
               goSub processLocalDimVar
            end

            for d1 = 1 to dim1
               if len( dim2 ) then
                  for d2 = 1 to dim2
                     dVar = debugger_Fetch( "GETLOCALDIMTYPE", varName, d1, d2 )
                     goSub processLocalDimVar
                  next
               end else
                  dVar = debugger_Fetch( "GETLOCALDIMTYPE", varName, d1 )
                  goSub processLocalDimVar
               end
            next

         case OTHERWISE$
            varData  = debugger_Fetch( "GETLOCAL", varName )
            goSub escapeVarData

            localVars := varName : @vm : varType : @vm : varData : @fm

      end case

   next

   localVars[-1,1] = ""

   swap    @fm with CRLF$ in localVars
   convert @vm to "|" in localVars

   osWrite localVars to LOCAL_DUMP_ID$

return
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
dumpLabelledCommonVars:
   lCommList  = debugger_Fetch( "LISTLABELLED" )
   lCommVars = ""

   xCount = count( lCommList, @fm ) + ( lCommList # "" )
   for x = 1 to xCount

      var     = lCommList<x>

      commName = var[1,@vm]
      varName  = var[col2()+1,@vm]
      varType  = var[col2()+1,@vm]
      varDim   = var[col2()+1,@vm]

      begin case
         case ( varType = VAR_TYPE_DESC_UNASSIGNED$ )
            lCommVars := commName : @vm : varName : @vm : varType : @vm : "" : @fm

         case ( varType = VAR_TYPE_DESC_IDISPATCH$ )
```

```
            lCommVars := commName : @vm : varName : @vm : varType : @vm : "" : @fm

         case ( varType = VAR_TYPE_DESC_DIM_ARRAY$ )
            lCommVars := commName : @vm : varName : @vm : varType : @vm : "" : @fm

            * // We need to iterate across the matrix ...
            dim1 = varDim[1,@svm]
            dim2 = varDim[col2()+1,@svm]

            d1 = 0
            d2 = 0
            if len( dim2 ) then
                dVar = debugger_Fetch( "GETLABELLEDDIMTYPE", commName, varName, d1, d2 )
                goSub processLabelledCommonDimVar
            end else
               dVar = debugger_Fetch( "GETLABELLEDDIMTYPE", commName, varName, d1 )
               goSub processLabelledCommonDimVar
            end

            for d1 = 1 to dim1
               if len( dim2 ) then
                  for d2 = 1 to dim2
                     dVar = debugger_Fetch( "GETLABELLEDDIMTYPE", commName, varName, d1, d2 )
                     goSub processLabelledCommonDimVar
                  next
               end else
                  dVar = debugger_Fetch( "GETLABELLEDDIMTYPE", commName, varName, d1 )
                  goSub processLabelledCommonDimVar
               end
            next

         case OTHERWISE$
            varData  = debugger_Fetch( "GETLABELLED", commName, varName )
            goSub escapeVarData

            lCommVars := commName : @vm : varName : @vm : varType : @vm : varData : @fm

      end case

   next

   lCommVars[-1,1] = ""

   swap @fm with CRLF$ in lCommVars
   convert @vm to "|" in lCommVars
   osWrite lCommVars to LCOMM_DUMP_ID$

return
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
processLocalDimVar:

   dVarType = dVar<1>
   varData = ""

   begin case
      case ( dVarType = VAR_TYPE_DESC_UNASSIGNED$ )
         null

      case ( dVarType = VAR_TYPE_DESC_IDISPATCH$ )
         null

      case ( dVarType = VAR_TYPE_DESC_DIM_ARRAY$ )
         * // This HAS to be an error!
         null

      case OTHERWISE$
         varData = debugger_Fetch( "GETLOCAL", varName, d1, d2 )
         goSub escapeVarData

   end case
```

```
   dVarName = varName : "(" : d1
   if len( dim2 ) then
      dVarName := "," : d2
   end
   dVarName := ")"

   localVars : = "   " : dVarName : @vm : dVarType : @vm : varData : @fm

return
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
processLabelledCommonDimVar:

   dVarType = dVar<1>
   varData = ""

   begin case
      case ( dVarType = VAR_TYPE_DESC_UNASSIGNED$ )
         null

      case ( dVarType = VAR_TYPE_DESC_IDISPATCH$ )
         null

      case ( dVarType = VAR_TYPE_DESC_DIM_ARRAY$ )
         * // This HAS to be an error!
         null

      case OTHERWISE$
         varData = debugger_Fetch( "GETLABELLED", commName, varName, d1, d2 )
         goSub escapeVarData

   end case

   dVarName = varName : "(" : d1
   if len( dim2 ) then
      dVarName := "," : d2
   end
   dVarName := ")"

   lCommVars := "   " : commName : @vm : dVarName : @vm : dVarType : @vm : varData : @fm

return
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
escapeVarData:
  swap "\"  with "\\"   in varData
   swap @rm  with "\xFF" in varData
   swap @fm  with "\xFE" in varData
   swap @vm  with "\xFD" in varData
   swap @svm with "\xFC" in varData
   swap @tm  with "\xFB" in varData
   swap \09\ with "\t"   in varData
   swap \0A\ with "\n"   in varData
   swap \0D\ with "\r"   in varData
   swap \00\ with "\0"   in varData
return
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
```